VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Malba štětcem v 2D rastrové grafice

Brush Painting in 2D Raster Graphics

2010 Lukáš Tvrdý

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave 7. mája 2010 .....

Rád by som poďakoval Ing. Petru Gajdošovi PhD. za rady a pomoc pri tvorbe diplomovej práce. Ďakujem tímu programátorov open-source projektu Krita, ktorí mi pomáhali pri integrovaní štetcov do programu Krita a umelcom, ktorí testovali štetce a vytvorili krásne maľby.

Abstrakt

Digitálne štetce sú základným nástrojom každého umelca digitálneho umenia. Cieľom práce je vytvoriť štetce, pomocou ktorých budú umelci efektívne vytvárať digitálne umenie. Štetce, ktoré sme implementovali, nesimulujú maľbu tradičných štetcov pomocou fyzikálne korektných modelov. Umožňujú vytvárať širokú škálu použiteľných efektov pre tvorbu malieb, skíc, komiksov a textúr. Štetce simulujú kresbu uhľom, maľbu v štýle Sumi-e pomocou čínskeho štetca alebo striekanie s nástrojom Airbrush. Všetky štetce pracujú v reálnom čase so skvelou odozvou aj na väčších plátnach. Atribúty štetcov sú dynamické a môžu byť mapované na parametre grafického tabletu ako je tlak, sklon alebo rotácia. Štetce boli od začiatku implementované a integrované do komplexného nástroja pre digitálne maľovanie v open-source projekte Krita a boli používané umelcami.

Kľúčové slová: maľba štetcom, kresliace algoritmy, rastrová grafika, nefotorealistické zobrazenie

Abstract

Digital brushes are the basic tool every digital artist uses. The goal of this diploma thesis is to create brushes which can be used by artists to create their digital art more effectively. The implemented brushes do not simulate the painting of traditional art materials using physically correct models. They allow to create a wide range of effects that are useful in the creation of paintings, sketches, comics and textures. These brushes can simulate sketching with charcoal, painting in the Sumi-e style using a Chinese brush or spraying with an airbrush. All brushes work in real-time with excellent responsiveness, even on large canvases. The attributes of the brushes are dynamic, they can be mapped to the parameters of a graphics tablet like pressure, tilt or rotation. The brushes were from the beginning implemented and integrated into a complex tool for digital painting, the open-source project Krita and have already been used by artists.

Keywords: brush painting, painting algorithms, raster graphics, non-photorealistic rendering

Zoznam použitých skratiek a symbolov

NPR	–	Non-photorealistic Rendering
RLE	–	Run-length encoding
PNG	–	Portable Network Graphics
XML	–	Extensible Markup Language

Obsah

Zoznam tabuliek

Zoznam obrázkov

1 Úvod

Popri tradičnom umení zažíva rozvoj digitálne umenie. Digitálne technológie umožňujú maľovať, kresliť a vytvárať nové druhy umenia. Umelci dnes čoraz častejšie siahajú po použití digitálnych nástrojov nielen pre ich ľahkú dostupnosť. Z tradičných umelcov sa stávajú umelci digitálneho umenia. Digitálne štetce sú základným nástrojom každého digitálneho umelca.

Digitálny štetec vnímame ako nástroj, ktorý vytvára alebo spracúva obraz pomocou ťahov kontrolovaných voľným pohybom ruky cez vstupné zariadenie. Vstupné zariadenie je počítačová myš alebo grafický tablet. Cieľ tejto práce je navrhnúť a implementovať digitálne štetce rastrovej grafiky, pomocou ktorých môžu umelci digitálneho umenia vytvárať umenie efektívne. V rámci digitálneho umenia sa zameriavame na tvorbu malieb, skíc, komiksov a textúr. Nadväzujeme na prácu iných autorov digitálnych štetcov. Nesnažíme sa verne simulovať tradičné médiá pomocou fyzikálne korektných modelov. Chceme vytvoriť digitálne štetce, ktoré poskytujú širokú škálu efektov a ktoré poskytujú rýchlu odozvu v reálnom čase. Snahou je, aby štetce boli pre digitálnych umelcov použiteľné v praxi.

Niektoré digitálne štetce verne simulujú maľbu skutočnými štetcami ako je napríklad maľba akvarelovými farbami. Iné digitálne štetce poskytujú nové možnosti maľby, ktoré sú možné len vďaka počítačovej simulácii. Vytvárajú syntetické obrazy, ktoré zobrazujú skutočnosť abstraktným spôsobom, pomocou rôznych umeleckých štýlov a zámerne zanedbávajú detaily. Tieto metódy zobrazenia skutočnosti patria do oblasti nefotorealistických zobrazovacích metód. Digitálne štetce sú užitočné vo filmovom priemysle, nachádzajú použitie vo viacerých odvetviach grafického dizajnu alebo pomáhajú pri tvorbe abstraktných ilustrácií v encyklopédiách a v komiksoch.

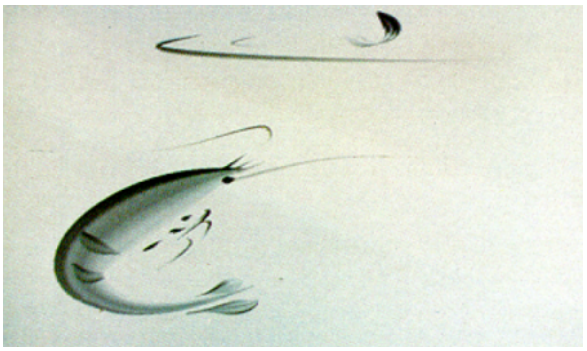
V prvej časti diplomovej práce popisujeme prácu autorov, ktorí vytvorili rôzne modely digitálnych štetcov, a ktorými sme sa inšpirovali pri návrhu vlastných štetcov. Stručne popisujeme oblasť nefotorealistických zobrazovacích metód počítačovej grafiky, kam patria digitálne štetce a opisujeme oblasti použitia štetcov. V ďalšej časti popisujeme štetce, ktoré sme navrhli a implementovali. V poslednej časti popisujeme prostredie programu Krita, do ktorého sme štetce integrovali a testujeme rýchlosť rôznych konfigurácií štetcov v tomto prostredí.

2 Digitálne štetce v praxi

2.1 Súčasný stav

Technológia digitálneho maľovania sa datuje po roku 1960. Prvý implementovaný užívateľský softvér sa datuje začiatkom skorých osemdesiatych rokov devätnásteho storočia. Prvý komerčný úspech digitálneho maľovania je program s názvom *Paint* a bol vytvorený pre štúdiá Lucasfilms. Ako prvý podporoval 32-bitový farebný priestor RGBA. Bol použitý vo filme *Star Trek II: The Wrath of Khan* na matte painting, teda na maskovanie scén. Bolo to v roku 1982. Tieto fakty spomína Alvy Ray Smith vo svojej publikácií [8] o histórií digitálneho maľovania. V nej popisuje históriu od dôb 8-bitových editorov, cez vývoj frame bufferu až po nástroj firmy Adobe, Photoshop. Alvy Ray Smith pracoval v roku 1975 na jednom z prvých kresliacich programov s názvom *SuperPaint*, za ktorý v roku 1990 dostal cenu *ACM SIGGRAPH Computer Graphics Achievement Award*. Jeho hlavným prínosom pre počítačovú grafiku bolo vynájdenie farebného priestoru HSV a podieľal sa na vynájdení konceptu alfa kanálu. Je jedným zo spoluzakladateľov slávneho štúdia Pixar [9].

Turner Whitted popisuje vo svojom článku [11] algoritmus pre kreslenie čiar bez aliasingu. Čiary vykresľuje pomocou masky štetca, ktorá je ťahaná po dráhe čiary. V každom bode čiary je vykreslený statický obraz, maska štetca bez aliasingu. Autor ďalej popisuje postup, ako vykresľovať 3D krivky. Používa na ich kreslenie techniku Z-buffer, inšpiroval sa Alvy Smithom a ďalej jeho prácu rozširuje. Pri kompozícii pixlov štetca a plátna berie algoritmus do úvahy aj hĺbku jednotlivých pixlov.



Obrázok 1: Strassmannov čínsky štetec

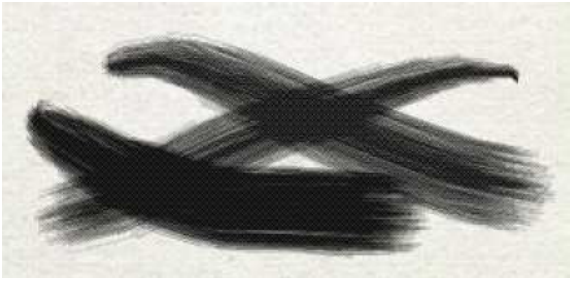
Jedným z pionierov v oblasti digitálnych štetcov je Steve Strassmann. Jeho práca s názvom *Hairy Brushes* [10] o simulácii čínskeho štetca tzv. *sumi-e* štýlu z roku 1986 je citovaná viac než 200-krát v rôznych publikáciách. Ide o simuláciu čínskeho štetca 1. generácie. Steve vo svojej práci popisuje pokročilejší model štetca, ktorý pozostáva zo štyroch základných modelov: štetec, ťah štetca, tvar štetca a papier. Štetec modeluje ako množinu štetín. Ťah štetca je modelovaný ako trajektória pozície a tlaku. Tvar štetca popisuje aplikáciu stavu štetca na plátno. Zahrňuje počiatočný stav štetca - rozloženie

štetín. Papier mapuje výsledok na zobrazovacie zariadenie. Táto vrstva umožňuje vytvárať obrazy so širokou škálou rozlíšení výsledného obrazu. Modulárny návrh štetca autor zvolil aj kvôli implementačnému prostriedku, ktorým bol objektovo orientovaný jazyk Zetalisp. Návrh umožnil Strassmannovi experimentovať so stochastickými modelmi toku a zmien farieb. Strassmann svoj systém implementoval ako neinteraktívny systém. Užívateľ definoval body, cez ktoré bude viesť ťah štetca, definoval v bodoch tlak a následne mohol generovať výsledný obraz. Neinteraktivita bola zvolená najmä kvôli rýchlosti vtedajšieho hardvéru. Jeden ťah štetca algoritmus vykresľoval minútu až dve. Zároveň neinteraktivita umožnila dosiahnuť efekty, ktoré sú v porovnaní s klasickým interaktívnym maľovaním, takmer nemožné.

Obrázok 2: Paul Haeberli: Paint By Numbers, impresionizmus pomocou digitálneho štetca a fotografie

Paul Haeberli pracoval ako výskumník v Silicon Graphics a venoval sa vývoju v oblasti počítačovej grafiky. Svoj výskum publikoval na svojej webovej stránke [4]. Jeho článok *Paint By Numbers: Abstract Image Representations* patrí ku kľúčovým článkom a značne inšpiroval vývoj v oblasti nerealistických zobrazovacích metód. Svedčí o tom i veľké množstvo citácií tejto práce. Vo svojom článku popisuje snahu o simulovanie umeleckého štýlu, ktorý sa volá impresionizmus. Ide o umelecký smer z 19. storočia, ktorý je charakteristický viditeľnými ťahmi štetca, otvorenou kompozíciou. Kladie dôraz na osvetlenie a jeho zmeny v priebehu času. Cieľom jeho práce bolo pretransformovať umelú alebo prírodnú scénu na abstraktný impresionistický obraz ako môžeme vidieť na obrázku 2. Celý proces transformácie ovláda užívateľ interaktívne pomocou ťahov štetcov. V jednotlivých ťahoch štetcov je použitá farba z obrazových bodov zdrojového obrazu. Následne je vykreslený ťah štetca s danou farbou. Paul pomocou vzorkovania zdrojového obrazu rieši problém *„put-that-color-here“*, ktorý sa vyskytuje v konvenčných nástrojoch pre digitálne maľovanie. Ide o procedúru, v ktorej užívateľ vyberie farbu z palety a vytvára ťah štetca. Tá bráni vytváraniu obrazov so širokým spektrom farieb, ktoré sú prítomné v klasických maľbách. Proces zmeny farby štetca je totiž pomalý. Každý

ťah štetca v Haeberliho systéme je popísaný atribútmi ako pozícia, farba, veľkosť ťahu, smer a tvar štetca. Tieto atribúty sú následne ukladané do súboru. Celá maľba je reprezentovaná množinou týchto ťahov. Maľba môže byť po uložení ďalej spracovaná, napríklad ťahy môžu byť zväčšované alebo otáčané. Paul popisuje i ďalšie možnosti tvorby výsledného obrazu. V časti, ktorá inšpirovala odbor počítačovej grafiky *Painterly rendering*, hovorí Paul o možnosti využitia 3D scény. V nej má štetec prístup k farbe, normále plochy a hĺbke bodu. Normála plochy môže byť použitá na riadenie ťahov štetcov. Paul prispel v rámci digitálneho maľovania aj aplikáciou *Dynadraw*, ktorej implementácia je voľne dostupná na jeho webovej stránke [4]. Ide o kresliacu techniku, v ktorej na pozície myši aplikujeme jednoduchý filter. Štetec je modelovaný ako fyzický objekt, ktorý má hmotnosť, rýchlosť a trenie. Myš akoby tlačí na štetec gumičkou. Užívateľ môže meniť hmotnosť a trenie, a tým dosahuje hladké, kaligrafické ťahy štetca.



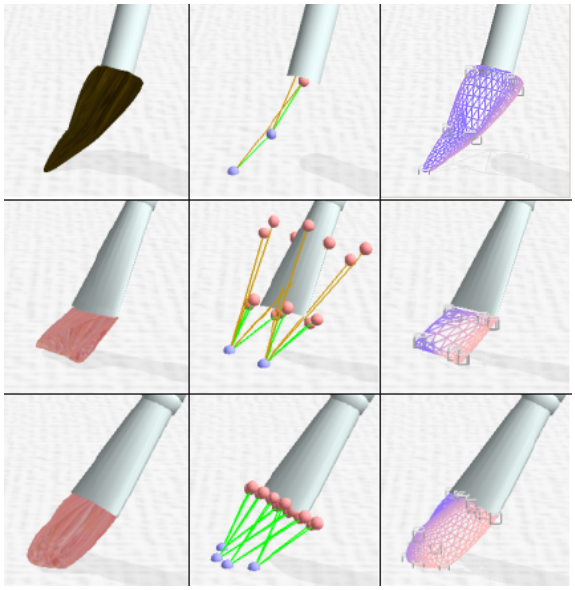
Obrázok 3: Clara Chen: ťahy čínskeho štetca

Clara Chen popisuje vo svojej práci [5] metódy na vytváranie čínskych malieb. Popisuje dve metódy dosiahnutia tohto cieľa. Pomocou existujúceho softvéru vytvára animáciu a v druhej metóde predstavuje a popisuje implementáciu jednoduchého modelu interaktívneho čínskeho štetca. Model štetca je tvorený oproti Strassmannovmu modelu 2D množina štetín. Navrhovaný systém je interaktívny, štetiny v modeli štetca reagujú na tlak tabletu a vytvárajú charakteristickú stopu čínskeho štetca (obrázok 3). Model zahŕňa i distribúciu farby medzi štetinami počas ťahu.

V diplomovej práci Erika Jansson popisuje algoritmy na maľovanie pomocou digitálnych štetcov. Vychádza z produktov firmy Adobe, konkrétne popisuje fungovanie štetcov a užívateľské rozhranie Adobe Photoshop a Corel Painter. Jej práca popisuje rôzne kompozície farby, zahŕňa i analýzu užívateľského rozhrania z pohľadu použiteľnosti. Množina štetcov, ktoré autorka navrhla, zahŕňa i štetce určené na modifikáciu fotografií. Napr. štetec, ktorý osvetľuje a stmieva vstupný obraz, štetec ktorý rozmazáva obraz. Zahŕňa i štetec, ktorý simuluje proces, v ktorom užívateľ robí prstom šmuhy na plátne. Popisuje i možnosti dynamickej zmeny farieb a tvaru štetca.

Bill Baxter je dnes jedným z najpoprednejších výskumníkom v oblasti digitálnej maľby. Vo svojom článku [1] popisuje komplexný systém na maľovanie, ktorý je pre umelca

veľmi rýchlo použiteľný. Hoci produkt Corel Painter vie generovať veľmi realistické ťahy štetca pomocou textúr a kompozitných trikov, Baxter ho označil za príliš zložitý a nákladný na zvládnutie. Vlastný systém, ktorý navrhol, má cieľ poskytnúť intuitívne prostredie na maľovanie, ktoré vyžaduje minimálne úsilie na zvládnutie. Momentálne pracuje na podobnom systéme s názvom Gustav pre firmu Microsoft[1]. Jeho práca zahŕňa dotykové užívateľské rozhranie. Model štetca tvorí 3D model, ktorý umožňuje tvoriť komplexné a pomerne fyzicky precízne ťahy štetca. O vizuálnu stránku výsledku sa stará vyvinutý obojsmerný dvojvrstvový model maľby, ktorý umožňuje vytvárať zaujímavé efekty pri kompozícií štetca s plátnom. Na maľbu používa hardvérový produkt, dotykové zariadenie podobné štetcu, ktoré navyše poskytuje spätnú väzbu pri maľovaní, čím dosahuje iný pocit z maľby ako pri použití grafického tabletu. Jeho systém je použiteľný i s grafickým tabletom, ktorý poskytuje minimálne 5 stupňov voľnosti. Síce jeho štetec patrí do inej kategórie ako štetce, o ktoré sa snažíme v tejto práca, v mnohom sa môžeme inšpirovať v procese maľovania z pohľadu užívateľského prostredia alebo v oblasti kompozície masky štetca s plátnom.



Obrázok 4: Bill Baxter: komplexný 3D model štetca

2.2 Postavenie štetcov v NPR

Digitálne maľovanie patrí medzi metódy počítačovej grafiky, ktoré sa označujú termínom *nefotorealistické zobrazenie*. V anglickej literatúre sa používa skratka NPR (z angl. *Non-*

[1]Realistický systém pre simuláciu maľovania `http://research.microsoft.com/en-us/projects/gustav/`

photorealistic rendering). Fotorealistické zobrazenie označuje metódy, ktoré sa snažia zobraziť scénu verne a realisticky, tak ako fotografia. Príkladom je zobrazovacia technika ray-tracing, ktorá dosahuje veľmi vysoký stupeň realistickosti zobrazenia. Nefotorealistické metódy sa zámerne od skutočnosti odkláňajú. Nepatria sem metódy, ktoré realistickosť nedosahujú z nejakého dôvodu. Dôraz sa kladie na určitý umelecký zážitok zo zobrazenia. Metódy NPR dodávajú scéne atmosféru a štýl. Cieľom je komunikovať vizuálnu informáciu čo najpresnejšie a tento cieľ dosahujú abstrahovaním reality. Fotorealizmus je v určitých prípadoch neprehľadný. V zložitej scéne sa ťažko vyznáme. Zanedbaním nepodstatných detailov a zdôraznením dôležitých častí informáciu zo scény lepšie komunikujeme. Príkladom sú technické ilustrácie, orientačné mapy alebo anatomické atlasy, ktoré obsahujú zložité štruktúry a sú lepšie čitateľné použitím zobrazovacích techník NPR. Scény môžu obsahovať často i dodatočné informácie o vzhľade objektov ako je tlak, teplota alebo rýchlosť. Dôležitý problém v oblasti počítačovej grafiky je vizualizácia veľkého a komplexného množstva informácií a práve tu techniky NPR nachádzajú svoje uplatnenie.

Pri tvorbe animovaných filmov je použitie NPR techník nevyhnutné. Zložité scény sa vytvárajú na počítači a tie sa následne kombinujú s ručne kreslenými objektami. Realisticky zobrazené scény v kombinácii s ručne maľovanými objektami by pôsobili rušivo. NPR rozšírila algoritmy počítačovej grafiky o osvetľovacie metódy (cel-shading, toon shading), ktoré generujú obrazy, ktoré vyzerajú ako ilustrácie. Umožňujú tvoriť dokonalé kompozície so spomínanými ručne maľovanými objektami.

Digitálne maľovanie v NPR rozdeľujeme do dvoch kategórií. V prvej kategórii sa nachádzajú interaktívne metódy. V nich užívateľ aplikuje ťahy štetca pomocou klasického vstupného zariadenia (tablet, myš) a úlohou algoritmov je generovanie výsledného obrazu s cieľom simulovať určitý tradičný umelecký materiál. Napríklad výsledkom sú ťahy simulujúce olejomaľbu, vodové farby, pastely alebo maľbu uhľom.

V druhej kategórii sa nachádzajú polo-automatické až automatické metódy maľovania. Úžívateľ v tomto prípade poskytne vstupné dáta ako je napr. 2D obraz alebo 3D scéna, prípadne 2.5D obraz. 2.5D obraz obsahuje dodatočné informácie o obrazových bodoch, napríklad hĺbku. Počítač na základe analýzy dát určí zaujímavé oblasti a vygeneruje ťahy štetcov s premenlivou veľkosťou, farbou a inými atribútami, ktoré charakterizujú ťah štetca. Tento proces je buď plne automatický alebo užívateľ pomáha pri výbere zaujímavých častí scén alebo volí štýl maľby. V 2D obrazoch sa používajú algoritmy na spracovanie obrazu ako je napr. detekcia hrán. Pri 3D scénach sa analyzuje geometria a tá určuje výsledné ťahy štetcov.

2.3 Použitie digitálnych štetcov v praxi

Matte painting

Názov tejto oblasti by sa dal preložiť ako maskovacie maľovanie. Takéto maľovanie pri tvorbe filmov pomáha pri tvorbe rôznych scén, ktoré nie je možné z rôznych dôvodov natočiť. Ide o fantastické svety, o scény, ktorých stavba by bola drahá alebo o miesta, ktoré nie je možné navštíviť alebo vytvoriť. Pôvodne sa časť scény ručne maľovala na sklo, ktoré bolo umiestnené pred kamerou. Dnes sa používa digitálne spracovanie, a tu nastupujú digitálne štetce. Digitálne spracovanie sa používa už od polovice 80-tých rokov 20. storočia. Vo filme Young Sherlock Holmes bola scéna, v ktorej bola použitá technika matte paintingu a bol použitý softvér Pixar zo štúdií LucasFilm [12]. Digitálne štetce sa používajú najmä v procese návrhu scény a taktiež v post-produkcii scény. Umelec vo fáze návrhu kombinuje fotografický referenčný materiál, rôzne 3D modely a ťahy digitálnych štetcov, ktoré vznikajú použitím tabletu tak, aby zachytil čo najlepšie požadovanú atmosféru. Kreslenie je na celom procese najdôležitejšie, pretože spája použité elementy do jedného celku. Vo fáze post-produkcie sa dokresľujú rôzne ďalšie elementy ako je napr. dážď, sneh alebo hmla. Umelci prevažne používajú bežné tzv. klasické štetce na maľovanie krajiny. Špeciálnymi štetcami dotvárajú detaily ako je napr. spomenutý dážď. Špeciálnym štetcom viacmenej vytvárajú rôzne zaujímavé textúry. Tie môžu mať syntetický charakter v prípade, keď ide napr. o animovaný film, alebo dosahujú fotorealistickú kvalitu. Syntetický charakter textúry a detailov si môžeme všimnúť v práci Davida Revoya na projekte Durian. Fotorealistická kvalita je požadovaná v hraných filmoch, príkladom je oskarový film z roku 2008 The Curios Case Of Benjamin Button[2]. S matte paintingom súvisí i tzv garbage matte (maskovanie smetia). Kreslič ručne štetcami v post-produkcii maže z obrazu „smetie“ v podobe rôznych stojanov na osvetlenie, mikrofónov a podobne.

Obrázok 5: David Revoy: matte painting,koncept pre film Sintel

[2]Ukážky matte paintingu z tohto filmu môžeme vidieť na `http://www.matteworld.com/film/2008/button_2.html`

Koncept art

Tento pojem označuje druh umenia, ktorý vznikol v súvislosti s dizajnom automobilov. Ide o druh ilustrácie, ktorá zachytáva myšlienku, atmosféru alebo štýl. Ide o návrh, ktorý odráža základné myšlienky finálneho produktu. Finálny produkt môže byť scéna počítačovej hry alebo hraného alebo animovaného filmu. Koncept umelci vytvárajú tieto ilustrácie prevažne použitím softvéru typu Adobe Photoshop alebo Corel Painter, z ktorých používajú rôzne štetce. Efektivitu zvyšujú použitím grafických tabletov. Ich workflow môže vyzerať i tak, že ilustráciu vytvoria pastelkami, akrylovými alebo olejovými farbami alebo inými klasickými kresliacimi nástrojmi a naskenované obrazy neskôr upravujú digitálnymi filtrami a štetcami. Tradičné nástroje sa im snažia vynahrádzať v čo najväčšej miere softvérové nástroje. S koncept artom súvisí tvorba storyboardu. Storyboard je návrhový artefakt, ktorý sa používa pri tvorbe rôznych animácií, filmov, webových alebo počítačových hier. Slúži na to, aby autor príbehu mohol vyjadriť a komunikovať svoju myšlienku tak, aby ju pochopili ostatní spolupracovníci, ktorý ju budú realizovať. Ide o hrubý náčrt scény, ktorá je zachytená graficky zorganizovanými náčrtmi v časovej následnosti. Najdôležitejšie je vyjadriť pomocou storyboardu odpovede na otázky kto, čo, kedy, kde a prečo. Storyboard je maľovaný ručne alebo pomocou digitálnych štetcov.

Obrázok 6: David Revoy: koncept art pre animovaný film Blender Foundation - Sintel

Web-design

V rámci web-designu sú digitálne štetce používané vo veľkej miere. Či už ako korekčný nástroj alebo dekoratívny nástroj. Pomocou štetcov vytvárajú dizajnéri grafické elementy, ktoré spájajú do celkov, prípadne vytvárajú pestrú ilustráciu. Súčasťou tvorby návrhu dizajnu profesionálnej webovej stránky môže byť koncept, náčrt webovej stránky. Pre návrh dynamického a interaktívneho webového obsahu sa vytvára, aj pomocou digitálnych štetcov, už spomínaný storyboard. Štetce sa používajú aj na úpravu fotografií pre web. Ide o retušovanie, vytváranie nerealistických snímkov, tzv. foto manipulácia. Upravené a retušované fotografie sú využívané na weboch rôznych magazínov a objavujú sa aj v tlačenej verzii.

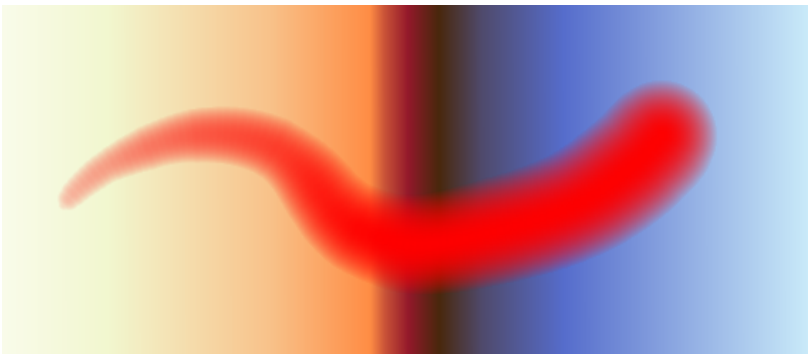
Pixel art

Rastrová grafika je tvorená pomocou množiny pixlov (obrazových bodov), ktoré sú uložené v digitálnom obrázku v pravidelnej mriežke. Pixel art je druh umenia, v ktorom umelec vytvára dielo na úrovni pixlov. Úroveň pixlov dosahuje pri veľkom zväčšení digitálneho obrazu v obľúbenom grafickom editore. Autori týchto diel sú navyše obmedzení technológiou zobrazenia. Kedysi sa pixel art vytváral na starých počítačoch, ktoré dosahovali nízku bitovú hĺbku zobrazenia (počítače ako Atari, Amiga...). Dnes pixel art nachádza uplatnenie napríklad v aplikáciách na mobilných telefónoch alebo PDA zariadeniach. V praxi sa najviac s pixel artom stretávame v podobne ikon na ploche operačného systému s grafickým desktopovým prostredím (Windows, KDE, GNOME). Pixel art nie je náročný na použité nástroje v kresliacom softvéri. Postačí bežný digitálny štetec prítomný už v jednoduchých aplikáciách ako je Maľovanie v MS Windows.

3 Implementované digitálne štetce

3.1 Soft brush

V každom pokročilejšom grafickom editore určenom pre digitálne maľovanie nájdeme digitálny štetec, ktorý patrí do kategórie štetcov pracujúcich na princípe označovanom v anglickej literatúre ako rubber-stamping. Štetec je reprezentovaný pomocným obrazom. Nazývame ho maska štetca. Maska štetca je v jednotlivých ťahoch štetca kombinovaná s obrazom, ktorý reprezentuje plátno. Operácia kombinácie sa označuje ako Bit BLIT (z anglického bit-block image transfer). Táto operácia kombinuje viacero obrazov do jedného výsledného obrazu pomocou určitej kompozitnej operácie. Kompozitná operácia môže byť napr. alfa-miešanie (anglicky alpha blending). Grafické editory väčšinou podporujú veľkú množinu kompozitných operácií. Program Krita, do ktorého integrujeme navrhované štetce, podporuje momentálne 25 rastrových operácií. Koncept alfa kanálu v počítačovej grafike sa využíva práve pri týchto kompozitných operáciách. Thomas Porter a Tom Duff popísali aritmetiku kompozitných operácií vo svojom článku [7] z roku 1984. Popisujú 12 základných operácií a ich inverznú verziu. Mnoho ďalších nových kompozitných operácií vychádza práve z týchto 12 základných operácií, pričom sa modifikujú funkcie, ktoré kombinujú obrazy do výsledného obrazu. Operácie sa často v praxi delia do rôznych kategórií ako napr. aritmetické operácie alebo operácie, ktoré modifikujú svetelnú zložku farieb. Na obrázku 7 vidíme alfa-miešanie Soft brush štetca s gradientom.



Obrázok 7: Kompozitná operácia alfa-miešanie

Štetec, ktorý bol implementovaný v Krite tak, aby pracoval na princípe rubber-stamping, bol pomenovaný Soft brush (v preklade mäkký štetec). V Krite už podobný štetec existuje a je pomenovaný názvom Pixel Brush, konkrétne v móde pomenovanom ako Auto brush. Motiváciou pre implementáciu bolo oboznámiť sa s problémami pri implementácii tohto typu štetca a zároveň priniesť niečo nové a pokúsiť sa o efektívnejší štetec. Efektívnejší po stránke rýchlosti a zároveň efektívnejší pri jeho nastavovaní a používaní.

Masku štetca môžeme určiť dvoma základnými spôsobmi. Procedurálne, kedy použijeme výpočet hodnoty jasu alebo alfa kanálu pre jednotlivé pixle alebo masku štetca tvorí

priamo digitálny obraz. Pokiaľ použijeme digitálny obraz, napríklad fotografiu uloženú v súbore, máme dve možnosti. Buď masku štetca vykresľujeme s farebnou informáciou obsiahnutou v obraze, alebo obraz konvertujeme na čiernobiely obraz a následne obraz interpretujeme ako alfa masku. Jednotlivé body obrazu reprezentujú hodnotu alfa kanálu. Biely pixel znamená úplnú priehľadnosť, čierny pixel znamená úplné krytie. Užívateľ pri tejto variante môže masku ľubovoľne zafarbovať. Zvolí si farbu, ktorou maska bude vykresľovaná. Pixle obrazu určujú len hodnotu alfa kanálu. Tento spôsob je veľmi často používaný digitálnymi umelcami. Svedčia o tom veľké zbierky tohto typu štetcov na webe. Rastrové programy používajú pre štetce vlastné súborové formáty. Program GIMP[3] používa formáty GIH (skratka Gimp Image Hose), GBR (Gimp Brush) a VBR (Variable brush). Súborové formáty okrem samotného digitálneho obrazu obsahujú aj ďalšie informácie ako je napríklad medzera (anglicky *spacing*), zmena mierky a ďalšie dynamické vlastnosti. VBR formát obsahuje nastavenie štetcového systému v textovej podobe. Rozborom súborového formátu štetcov programu Adobe Photoshop bolo zistené, že jeden súbor obsahuje množinu masiek štetcov uložených ako čiernobiely obraz komprimovaný pomocou RLE algoritmu. Okrem obrazu súbor obsahuje mnoho ďalších atribútov, ktoré konfigurujú štetcový systém a jeho dynamické vlastnosti.

Procedurálnym spôsobom definujeme tvar štetca tak, že zvolíme určité parametre. Určíme požadovaný tvar, zvyčajne je podporovaný tvar elipsy alebo štvoruholníka. Užívateľ definuje výšku a šírku zvoleného tvaru. Pokiaľ definuje výšku a šírku, často je vyžadované, aby užívateľské rozhranie umožňovalo uzamknúť pomer veľkostí medzi výškou a šírkou. Pokiaľ nastane zmena veľkosti výšky, šírka sa v aktuálnom pomere prepočíta a takisto to platí pre zmenu šírky. Alternatívny spôsob zadávania rozmerov zvoleného tvaru je kombinácia parametrov priemer a pomer veľkosti (anglicky *aspect ratio*). Tento spôsob je použitý v implementovanom štetci. Merná jednotka je vo väčšine prípadov pixel. Tvar ďalej môžeme ovplyvniť parametrami ako je zmena mierky a rotácia. Pre tieto zmeny používame odpovedajúce transformačné matice pre rotáciu a zmenu mierky.

Obrázok 8: Soft brush: procedurálna maska, priemer 100 px,rotáca 135 stupňov

[3]Dostupný na `http://www.gimp.org`

Rasterizácia masky štetca prebieha tak, že sa zo zvolených parametrov štetca spočíta ohraničujúci priestor masky (anglicky *bounding box*). Pri uhle rotácie 0 stupňov je priestor daný priamo rozmermi štetca, teda priemerom a pomerom strán. Pri odlišnej rotácii vytvárame obdĺžnikovú oblasť danú rozmermi štetca, následne oblasť rotujeme okolo stredu oblasti o zadaný uhol a spočítame ohraničujúci priestor, ktorý je daný súradnicami rohov obdĺžnikovej oblasti. Tento priestor je opäť obdĺžnikového alebo štvorcového tvaru. Následne si vytvoríme pomocný pixel buffer odpovedajúci rozmerom ohraničujúceho priestoru. Jedným z parametrov štetca je i aktuálne zvolená farba. V procese rasterizácie meníme hodnotu alfa kanálu zvolenej farby a tým tvoríme výsledný tvar masky. Pre hodnoty alfa kanálu používame určitú matematickú funkciu. Spočítame pozíciu pixlu v maske. Použijeme algoritmus, ktorý bude prechádzať jednotlivé riadky masky štetca a pre každý pixel spočítame hodnotu alfa kanálu na základe stredovej rovnice pre elipsu.

```
float centerX = width * 0.5 + subPixelX;
float centerY = height * 0.5 + subPixelY;

for (int y = 0; y < height; y++){
  for (int x = 0; x < width; x++){
      float maskX = x - centerX;
      float maskY = y - centerY;

      float result = pow(maskX/a,2) + pow(maskY/b,2);

      color.setOpacity(1.0 - result);
      setPixel(x,y, color);
  }
}
```

Výpis 1: Rasterizácia procedurálnej masky štetca

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1 \tag{1}$$

Vo výpise vidíme výpočet stredu masky. Premenné width a height reprezentujú šírku a výšku pixel bufferu. Premenné s prefixom subPixel sú reálne čísla v rozsahu (0,1). Pokiaľ program umožňuje používať funkciu zoom, kedy užívateľ môže náhľad na obraz ľubovoľne zväčšovať a zmenšovať a maľovať na obraz bez zmeny jeho veľkosti, mali by sme zahrnúť do výpočtu masky i presnosť na sub-pixel. Pri zobrazení plátna s veľkosťou zoomu 200% má pixel dvojnásobnú veľkosť. Súradnice, ktoré posiela zariadenie, musia byť transformované podľa hodnoty veľkosti zoomu. To znamená, že súradnice pozície, na ktorú máme vykresliť masku štetca sú reálne. Súradnice pre pozíciu pixel bufferu, ktorý bude následne zlúčený kompozitnou funkciou s obrazom plátna, sú však celé čísla. Preto zohľadňujeme presnosť na sub-pixel vo výpočte rasterizácie masky štetca zmenou priehľadnosti pixlu.

Pokiaľ je hodnota stredovej rovnice menšia alebo rovná 1, daný pixel patrí do zvoleného tvaru. Pre ostatné body nastavíme hodnotu alfa kanálu na úplne priehľadný pixel. Pixel vykreslíme so zvolenou farbou a nastavíme mu alfa kanál na hodnotu funkcie,

ktorú si zvolíme. Niektoré implementácie používajú funkciu podobnú Gaussovej funkcii ako napr. GIMP alebo sa používa i inverzná hodnota stredovej funkcie elipsy. V implementovanom štetci Soft brush užívateľ môže definovať vlastnú funkciu. Užívateľ definuje body, cez ktoré funkcia prechádza. Následne sa spočíta vektor hodnôt funkcie danej krivkou. Veľkosť vektora je daná polomerom masky a zároveň slúži i ako rozlíšenie, pri ktorom sú hodnoty funkcie vzorkované. Hodnotu alfa kanálu určuje výsledok stredovej rovnice elipsy, ktorý slúži ako index prvku vo vektore. Keďže ide o reálne číslo v intervale (0.0,1.0), index môžeme určiť dvoma spôsobmi. Výpočet mapovania jasu prevedieme metódou najbližší sused, použijeme zaokrúhľovanie.

Tento postup má nevýhody, výsledná maska štetca je zasiahnutá aliasom a je teda „zubatá“, užívateľ vidí jednotlivé prechody intenzity jasu farby. Alias vzniká kvôli tomu, že spojitú veličinu reprezentuje diskrétnou veličinou. Tento alias v implementácií odstraňujeme pomocou lineárnej interpolácie medzi viacerými prvkami vektoru. Výslednú hodnotu alfa kanálu určujeme medzi dvoma hodnotami vektora.

```
int getMappedOpacity(float result, int radius){
  int index = result * radius;
  float fraction = result - floor(result);

  float alpha = ((1.0 - fraction) * curveData[index] + fraction * curveData[index+1]);
  return OPACITY_MAX * alpha;
}
```

Výpis 2: Výpočet mapovania jasu lineárnou implementáciou

Alias môžeme taktiež odstrániť tak, že hodnotu alfa kanálu určíme priamo zo vzorca pre výpočet definovanej funkcie. Vektor hodnôt bol použitý kvôli výkonnosti. Iné riešenie používa program GIMP. Funkciu reprezentuje vektorom so štvornásobne väčším množstvom vzoriek hodnôt použitej funkcie ako je potrebné na výpočet masky. Vykonáva určitý super-sampling.

Užívateľom definovaná funkcia ovplyvňuje vlasnosť, ktorá sa označuje ako „mäkkosť“ štetca. Táto vlastnosť je označovaná v grafických editoroch niekedy ako tvrdosť (hardness) alebo mäkkosť (softness). V ostatných grafických editoroch sa mäkkosť udáva pomocou číselnej hodnoty, obdobný štetec v Krite používa hodnoty vertical a horizontal fade. Užívateľom definovaná krivka poskytuje mocnejší nástroj pri určovaní výslednej intenzity jednotlivých pixlov. Umožňuje navyše kontrolovať celkové krytie a nielen mäkkosť. Pomocou krivky dokáže užívateľ vytvárať omnoho zaujímavejšie masky štetca a i také, ktoré sa pomocou číselných hodnôt nedajú vytvoriť. Často sa pri vykresľovaní ťahov tabletom mapuje tlak tabletu na priehľadnosť a zmenu veľkosti. Zaujímavá možnosť a kontrola nad ťahom môže byť dosiahnutá i mapovaním tlaku tabletu na mäkkosť ťahu. Na obrázku 9 vidíme rozdiel medzi ťahom s dynamickou zmenou priehľadnosti a zmenou mäkkosti.

Obrázok 9: Soft brush: dynamická zmena priehľadnosti a mäkkosti

Do modelu štetca Soft brush bol pridaný ďalší parameter s názvom hustota (anglicky *density*). Tento parameter percentuálne určuje hustotu pixlov v zvolenom tvare. Napr. pri hustote 50% je polovica pixlov tvoriacich masku nahradená pixlom, ktorý je úplne priehľadný a tým vyvoláva dojem chýbajúceho bodu pri následnej kompozícii masky s plátnom. Tento efektu sa používa napr. na jednoduchú simuláciu kriedy, pastelky alebo kresby uhľom.

Obrázok 10: Soft brush: kresba uhľom, hustota 33%

Ďalšie podporované vlastnosti sú všeobecnejšieho charakteru. Vlastnosť, ktorá sa volá medzera (anglicky *spacing*), definuje ako ďaleko sa v ťahu štetca nachádzajú jednotlivé masky štetca. Užívateľ definuje veľkosť medzery percentuálne z priemeru zvoleného tvaru. Výsledná veľkosť je v jednotkách pixlov a je použitá pri vzorkovaní ťahu štetca. Ťah štetca je tvorený tak, že maska štetca je vykresľovaná v konštantných vzdialenostiach pozdĺž ťahu. Konštantný interval je udaný práve medzerou. Čím je medzera menšia, tým náročnejšie je vykresľovanie ťahu, pretože maska štetca sa musí rasterizovať viackrát ako pri vyšších hodnotách medzere. Pri nízkych hodnotách je ťah plynulý a pri vysokých hodnotách rozpoznávame jednotlivé masky štetca.

Ďalšia vlastnosť všeobecnejšieho charakteru sa nazýva trasenie (anglicky *jittering*). Taktiež definujeme vzdialenosť v pixloch percentuálne z polomeru zvoleného tvaru. Vzdia-

Obrázok 11: Soft brush: medzera 50% a 100%

lenosť je však použitá na vychyľovanie pozície masky štetca počas ťahu. Táto vlastnosť umožňuje simulovať trasenie ruky pri maľovaní. Využívame pri tom generátor náhodných čísel, pomocou ktorého generujeme náhodný vektor posunutia.

```
x = x + (2 * radius * random(0.0,1.0) - radius) * jitter ;
y = y + (2 * radius * random(0.0,1.0) - radius) * jitter ;
```

Výpis 3: Výpočet súradníc pri trasení

Funkcia random() vracia číslo v rozsahu (0,1). Polomer tvaru je v premennej radius. Jitter je reálne číslo, ktoré vyjadruje percentuálne veľkosť posunu. V implementácii v programe Krita je v rozsahu (0.0,5.0) pričom horná hranica znamená maximálny možný posun 500% z polomeru tvaru štetca.

Obrázok 12: Soft brush: trasenie s hodnotou 0.2 (20%)

S digitálnym maľovaním je spojené i použitie vstupných zariadení, ktoré poskytujú viac stupňov voľnosti ako je grafický tablet. Tablet v kombinácií s perom poskytuje informácie o tlaku, sklone , prípadne o rotácií. Tieto vlastnosti podporujeme i v navrhovaných štetcoch. Tieto vlastnosti označujeme ako *dynamické vlastnosti štetca*.

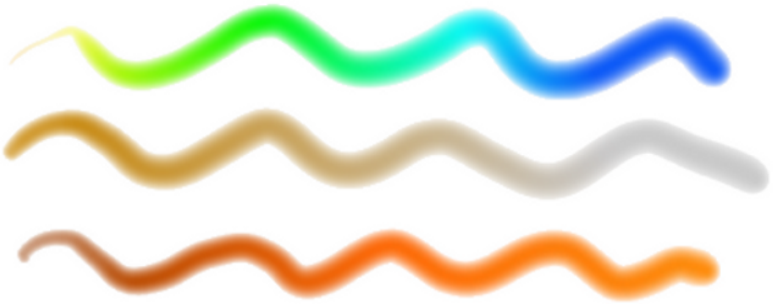
Zmena veľkosti tvaru je často dynamická vlastnosť. Tlak pera ovplyvňuje rozmery masky štetca. S väčším tlakom pri maľovaní vytvárame väčšiu masku štetca. Zmena tvaru prebieha tak, že je zachovaný pomer strán daného tvaru. Rotácia pera môže byť mapovaná na rotáciu masky štetca. To znamená, že pokiaľ je tvar štetca elipsa, užívateľ môže rotovať elipsou okolo jej stredu počas ťahu. Ďalšia podporovaná vlastnosť je dynamické krytie masky štetca. Tlak ovplyvňuje alfa kanál pri kompozícii masky štetca s plátnom. Je to veľmi používaná vlastnosť pri digitálnom maľovaní. Vlastnosti demonštruje obrázok 13.

Pokúsili sme sa doplniť tieto bežné vlastnosti o možnosť mapovať tlak pera na ďalšie vlastnosti farby. Zvolili sme si farebný model HSV.Tento model popisuje fyzikálne vnímanie farby. Používa k tomu atribúty odtieň farby (hue), sýtosť farby (saturation) a intenzitu farby (value). Odtieň farby vystihuje meno, ktoré je použité pre konkrétnu

Obrázok 13: Soft brush: dynamika veľkosti, rotácia tvaru a zmena krytia masky

farbu. Napríklad červená, modrá, zelená, žltá a podobne. Vo farebnom modeli je atribút odtieňa reprezentovaný pomocou uhla v rozsahu 0 až 360 stupňov. Sýtosť farby súvisí s jej priezračnosťou alebo čistotou vnímania. V modeli je atribút reprezentovaný percentuálne. 100% hodnota saturácie vyjadruje čiru farbu, pre menšie hodnoty farba bledne. Intenzita farby je taktiež vyjadrená percentuálne. Vyjadruje vlastnosti farby z pohľadu svetlosti a tmavosti. Užívateľ môže teda dynamicky v štetci Soft brush meniť odtieň farby, jej sýtosť a jas pomocou tlaku. Každú zložku farebného priestoru HSV môže kontrolovať samostatne. Navyše môže použiť implementované automatické znižovanie alebo zvyšovanie farby a tým simulovať zaujímavé efekty i bez tabletu. Automatické znižovanie alebo zvyšovanie funguje tak, že pri každom vykreslení masky zväčšíme počítadlo štetca, ktoré bolo inicializované na hodnotu 0. Užívateľ definuje maximálnu hodnotu počítadla. Môžeme ju nazvať dĺžka. Podiel hodnoty počítadla a dĺžky použijeme ako parameter transformácie zložky farebného modelu HSV. Podiel počítame v reálnych číslach. Získame hodnotu v intervale (0,1). Pokiaľ počítadlo prekročí hodnotu dĺžky, môžeme počítadlo nastaviť na východziu hodnotu, prípadne počítadlo ďalej nezvyšovať. Užívateľ môže nastaviť, či daný atribút rastie alebo klesá. Pri klesaní atribútu používame výsledok delenia so záporným znamienkom. Priebeh zmeny kontroluje užívateľ krivkou, čím dosiahne väčšiu kontrolu nad farebnou transformáciou. Efektívnejšie kontroluje zmenu vlastnosti tlak tabletu. Tlak tabletu posunieme do intervalu (0,1) a túto hodnotu použijeme na transformáciu zvolenej zložky HSV modelu. Tlak môžeme mapovať na interval (-1,1), a tým dosiahneme, že tlak zároveň kontroluje zväčšenie aj zmenšenie atribútu zároveň. Niektoré grafické tablety poskytujú informácie o sklone pera v osách X a Y. Túto informáciu môžeme použiť na kontrolu zmeny zložiek HSV modelu. Atribúty môžeme kontrolovať rôznymi ďalšími hodnotami ako je vygenerované náhodné číslo, uhol ťahu, rýchlosť ťahu a podobne.

Soft brush nachádza veľmi široké použite v praxi digitálneho maliara. Je vhodný pre tvorbu skíc, prvých náčrtov ale i na maľbu detailov. Používa sa takmer na všetko. Je to jednoduchý štetec, ale patrí k najpoužívanejším štetcom, aj pre jeho jednoduché ovládanie. Pomocou ďalších parametrov, ktorými sme štetec rozšírili môžeme, zefektívniť prácu pri digitálnom maľovaní. Krivka mäkkosti je efektívnejší spôsob nastavovania masky štetca ako poskytuje pôvodný štetec Pixel Brush v Krite. Pomocou parametra hustoty môžeme navyše modelovať médiá ako je uhoľ, pastelka alebo krieda. Dynamické zmeny vlastností farby štetca v modele HSV poskytuje zaujímavé riešenie problému *„put-*



Obrázok 14: Soft brush: dynamická zmena odtieňa,sýtosti a intenzity farby počas ťahu

that-color-there“, ktorý spomína Paul Haeberli v [3] a ktorý sme vysvetlili v časti 2.1. Tento štetec bol použitý pri tvorbe maľby, ktorú vidíme na obrázku 39 v prílohe.

3.2 Hairy brush

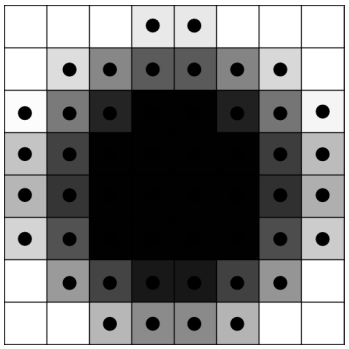
Odlišnú množinu štetcov tvoria digitálne štetce, ktoré sa snažia o simuláciu skutočných štetcov a nepoužívajú len statickú masku štetca použitú v štetci Soft brush, ktorú opakovane prilepujú na pozície ťahu. O prvý odlišný model sa snažil Steve Strassman, ako sme už spomínali v kapitole 2.1. Strassmannov štetec je však navrhnutý tak, že nefunguje v reálnom čase interaktívne. Strassmann bol vo svojej dobe ohraničený schopnosťami hardvéru. Existujú implementácie Strassmannovho štetca [4], ktoré už v reálnom čase fungujú, avšak iba pre malé plátno s rozmermi do stoviek pixlov. Preto som sa pokúsil o vlastný model štetca, ktorý pracuje v reálnom čase a bez limitu na veľkosť plátna.

Pôvodný názov vyvíjaného štetca bol *sumi-e*. Je to originálny názov pre čínsky štetec. Čínske štetce sa vyvinuli aj pre potreby kaligrafie. Výskum v oblasti simulácie tohto štetca pokračuje v Číne, kde sa snažia využiť simuláciu štetca pri tvorbe fontov [14]. Neskôr bol štetec premenovaný na *Hairy brush*, čiže vlasový štetec. Štetec totiž nesimuluje verne čínsky štetec, ako sa o to snažia iní autori [1]. Ide o všeobecnejší model štetca,jeho účel je skôr poskytnúť škálu zaujímavých efektov, ktoré pripomínajú štetce tvorené štetinami.

Pre štetec bol vytvorený jednoduchý model správania. Stopu štetca vytvára model častíc. Častice nazývame štetiny (anglicky *bristle*). Každá štetina má nasledovné vlastnosti: poloha, dĺžka, množstvo atramentu a farba. Poloha štetiny je vyjadrená v 2D súradniciach. Reálne čísla x a y vyjadrujú vzdialenosť štetiny od počiatku súradnej sústavy, a teda i od stredu masky štetca. Dĺžka štetiny je relatívna veličina, určujeme ju reálnym číslom v intervale (0,1). Množstvo tušu je tiež relatívne,vyjadrené reálnym

[4] Implementácia pomocou Java appletu je dostupná na `http://acg.media.mit.edu/people/golan/brush/`

číslom v intervale (0,1). Farba štetiny je reprezentovaná pixlom.



Obrázok 15: Hairy brush: model štetca, dĺžku štetín určuje intenzita a alfa kanál pixlu

Predtým než začneme vykresľovať ťah štetca, najprv vytvoríme na základe užívateľských parametrov počiatočný tvar štetca. Maska štetca slúži iba na určenie pozície štetiny v rámci tvaru. Užívateľ určuje tvar štetca, ktorý je tvorený množinou štetín. Štetinám určuje spomínané vlastnosti ako je dĺžka, poloha. Na určenie komplexnej množiny vlastností sme zvolili digitálny obraz. Digitálny obraz je získaný z procedurálnej masky alebo užívateľ priamo zvolí konkrétny obraz v užívateľskom rozhraní. Tento spôsob určenia tvaru je veľmi flexibilný, pretože umožňuje vytvárať zložité tvary jednoduchým spôsobom. Pozícia štetiny je daná vzdialenosťou od stredu dvojrozmernej kartézskej súradnej sústavy. Stred súradnej sústavy je položený v strede vstupného digitálneho obrazu. Stred obrazu je zvolený kvôli jednoduchosti, užívateľ nemusí definovať ďalší parameter. Zároveň sú veľmi časté masky štetca, ktoré majú tvar kruhu alebo elipsy. Stred súradnej sústavy však môže byť v ľubovoľnom mieste. Jednotlivé štetiny majú pridelené celočíselné súradnice.

```
int width = image.width();
int height = image.height();

int centerX = width * 0.5;
int centerY = height * 0.5;

for (int y = 0; y < height; y++){
  for (int x = 0; x < width; x++){
      bristle .setX(x - centerX);
      bristle .setY(y - centerY);
  }
}
```

Výpis 4: Výpočet súradníc štetín

Z dimenzie vstupného obrazu spočítame stred masky. V tomto prípade ignorujeme informáciu o sub-pixel pozícií v porovnaní so Soft brush štetcom, pretože rozmiestnenie štetín vytvárame v kroku, kedy táto informácia nie je dostupná a sub-pixel pozícia nie

je veľmi podstatná v našom modeli. Jednotlivé pozície štetín môžeme prípadne posunúť o sub-pixel posun pri prvom kontakte štetca s plátnom. Následne iterujeme obrazom a počítame súradnice každej štetiny v štetci. Množstvo atramentu v štetine je na začiatku nastavené na reálnu hodnotu 1, ktorá reprezentuje 100% množstvo atramentu.

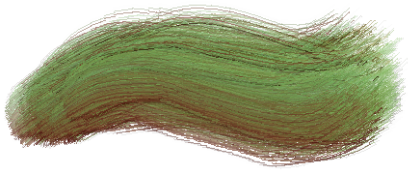
Dĺžku štetiny určujeme opäť z digitálneho obrazu. Konkrétne z jasu daného pixlu a hodnoty alfa kanálu pixlu (obrázok 15). Jas I určujeme vo farebnom obraze reprezentovanom vo farebnom priestore RGB pomocou vzorca 2.

$$I = R * 0.3 + G * 0.59 + B * 0.11 \tag{2}$$

```
for (int y = 0; y < image.height(); y++)
  for (int x = 0; x < image.width(); x++){
    color = image.getPixel(x,y);
    float length = ((255 − qGray(color)) ∗ qAlpha(color)) / 255;
    length /= 255.0;
  }
}
```

Výpis 5: Výpočet dĺžky štetiny z jasu a hodnoty alfa kanálu pixlu

Premenná *color* reprezentuje pixel vo farebnom priestore RGBA. Funkcia *qGray* vracia hodnotu jasu pixlu podľa vzorca 2. Funkcia *qAlpha* [5] vracia hodnotu alfa kanálu pixlu. Pixel, ktorý je úplne tmavý, má hodnotu 0. Táto hodnota však reprezentuje v našom modeli štetinu s maximálnou dĺžkou. Preto hodnotu obrátime. Pokiaľ pracujeme s digitálnym obrazom, pixel je uložený v 8-bitovom farebnom priestore, čiže maximálna hodnota jasu je 255. Hodnotu mapujeme na reálne číslo v intervale (0,1). Pokiaľ by sme chceli podporovať rôzne farebné priestory, vždy berieme intenzitu farby ako dĺžku štetiny. Do úvahy berieme alfa kanál. V prípade, že ide o pixel, ktorý je úplne priehľadný, štetinu nezaradíme do tvaru štetca. Týmto krokom teda môžeme vytvárať komplikované tvary štetca.



Obrázok 16: Hairy brush: farebná maska určuje štetinám farbu

Posledný atribút je farba. Užívateľovi poskytujeme dve možnosti. Buď zvolí masku štetca, ktorá je farebná a teda poskytuje farebnú informáciu. Farebný ťah vidíme na obrázku 16. Alebo užívateľ zvolí jednu farbu, ktorá bude použitá pre každú štetinu. Farebná maska je v tomto prípade použitá tak, že je prevedená na čiernobiely obraz a určuje

[5] Funkcie qGray a qAlpha sú súčasťou použitého frameworku Qt 4.6.x

len dĺžku štetín intenzitou jednotlivých pixlov.

Do modulu štetca boli pridané ďalšie parametre, ktoré kontrolujú počet štetín v maske štetca kvôli výkonu štetca. Už pre malé digitálne obrazy s rozmermi do stoviek pixlov získame veľké množstvo štetín, ktoré znižujú odozvu štetca. Každý pixel reprezentuje štetinu. V digitálnom maľovaní sú bežné veľké štetce s rozmermi do 1000 pixlov. Parameter, ktorý ovplyvňuje počet štetín, nazývame hustota (anglicky *density*). Užívateľ definuje aké percento štetín bude použité vo výslednom tvare štetca. Pri 100% hustote sú štetiny kladené jedna vedľa druhej vo vzdialenosti jedného pixlu. Pri znížení hustoty sú štetiny vyraďované pomocou generátora náhodných čísel s uniformnou distribúciou. To znamená, že každá štetina je vyraďovaná s rovnakou pravdepodobnosťou. Pri nastavení hustoty na 50% je polovica štetín vyradená. Tento parameter je časťou procesu predspracovania štetín. Pomocou neho je dosiahnutá ďalšia rôznorodosť stopy štetca. Pri ťahu štetca s hustými maskami veľa štetín prekresľuje rovnakú časť plátna a výsledok ťahu je veľmi podobný i s oveľa nižším počtom štetín ako vidíme na obrázku 17. Hustá maska štetca je taká, ktorá obsahuje veľmi malý počet pixlov, ktoré sú úplne priehľadné. Znížením počtu štetín v týchto prípadoch zlepšujeme odozvu štetca.

Obrázok 17: Hairy brush: hustota 100% štetín a 30% štetín.

Druhá možnosť pri ovplyvňovaní počtu štetín je priamo určiť ich počet. Opäť použijeme generátor náhodných čísel s uniformnou distribúciou. Užívateľ určí, aký počet štetín chce. Zo všetkých štetín, ktoré získame, následne vyberieme náhodne daný počet štetín. Výber štetín môže ďalej špecifikovať vlastnosť štetiny. Môžeme preferovať skôr dlhšie štetiny alebo štetiny, ktorých poloha od stredu masky štetca je nižšia. Preferovanie zaručíme tak, že štetiny podľa vlastnosti zoradíme vo vektore a vyberáme následne z časti vektoru, v ktorej štetiny s danou vlastnosťou dominujú.

Počas ťahu reálneho čínskeho štetca sa plátna dotýkajú iba niektoré štetiny. Túto skutočnosť taktiež modelujeme v štetci pomocou voliteľného parametru orezanie (anglicky *threshold*). Každá štetiny má parameter dĺžky. Pri vykresľovaní ťahu vykresľujeme len štetiny, ktoré sa aktuálne plátna dotýkajú. Test dotyku vykonávame pomocou tlaku, ktorý poskytuje tablet. Pokiaľ je štetina kratšia ako je aktuálny tlak, štetina je vyradená. Test prebieha v rámci každej úsečky, ktorá vzorkuje ťah. Orezanie vizuálne mení ťah štetca. Hrany štetca na obrázku 18 neobsahujú krátke štetiny a tým vzniká ťah, ktorý je tvrdší bez mäkkých prechodov jasu v porovnaní s ťahom bez orezania.

Obrázok 18: Hairy brush: ťah s orezaním krátkych štetín a bez orezania

Po vytvorení reprezentácie masky štetca môžeme začať vykresľovať ťah štetca. Väčšina štetcov v programe Krita využíva procedúru, ktorá vzorkuje ťah štetca. Keď užívateľ vstupným zariadením (počítačová myš alebo tablet) kreslí trajektóriu štetca, pričom sa mení poloha kurzoru na plátne, procedúra vytvára volanie funkcie, ktorá poskytuje parameter predchádzajúcej polohy a aktuálnej polohy. Tým vytvára pre štetec ťahy, ktoré sú vzorkované pomocou úsečiek. API programu Krita navyše umožňuje použiť procedúru, ktorá vzorkuje jednotlivé úsečky na pozície na základe užívateľsky definovanej vzdialenosti. Túto procedúru využíva štetec Soft brush. Vzdialenosť definuje pomocou parametra medzera. V štetci Hairy brush používame procedúru, ktorá vzorkuje celý ťah štetca len do úrovne úsečiek.

Vykresľovanie prebieha tak, že najprv si spočítame uhol úsečky ťahu s osou x. Všetky štetiny rotujeme uhlom, ktorý zviera úsečka ťahu s osou x. Využívame vlastnosť štetiny, jej poloha je daná v súradnicovej sústave masky, takže štetinu pred rotáciou neposúvame vzhľadom na pozíciu štetca v plátne. Rotáciou štetín uhlom úsečky dosiahneme to, že tvar štetca, a teda štetiny sledujú trajektóriu štetca. Rotácia môže byť voliteľný parameter, štetiny by sme nemuseli rotovať avšak ich rotáciou zvyšujeme realistickosť ťahu štetca.

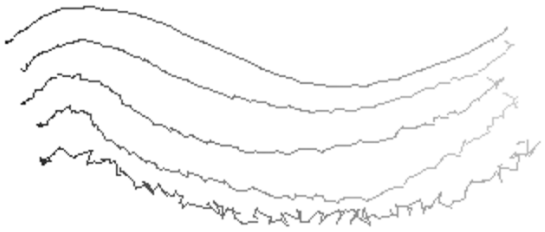
```
float dx = endX − startX;
float dy = endY − startY;

double angle = atan2(dy/dx);
```

Výpis 6: Výpočet uhla časti ťahu

Každú štetinu vykresľujeme samostatne, pretože má svoju vlastnú trajektóriu. Užívateľ trajektóriu štetín môže ovplyvňovať pomocou ďalších parametrov. Prvý parameter je náhodné posunutie štetiny (anglicky *random offset*). Pre štetinu vygenerujeme náhodné posunutie, pričom jeho veľkosť je obmedzená užívateľsky definovanou hodnotou. Pomocou tohto atribútu dosiahneme jednoduchú simuláciu nerovností na plátne, ktoré náhodne modifikujú trajektóriu štetiny. Tento efekt vidíme na obrázku 19, kde sme vykreslili dráhu jednej štetiny s rôznymi hodnotami náhodného posunutia. Pri vysokých

hodnotách dochádza k zväčšeniu stopy štetca.



Obrázok 19: Hairy brush: náhodné posunutie štetiny s hodnotami 0, 0.4, 1, 1.4, 2.0

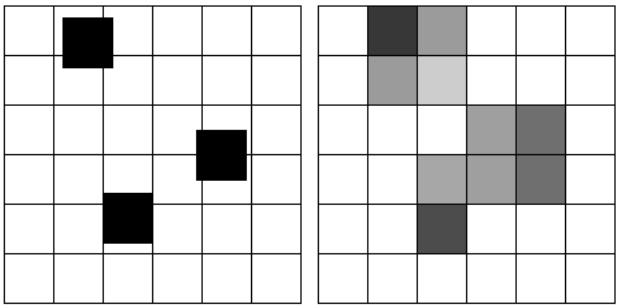
Druhý atribút definuje zmenu mierky. Zmena mierky pri štetcoch ako je napr. Soft brush, zväčšuje výsledný tvar štetca pričom sa mení maska štetca. Aj v prípade tohto štetca chceme, aby dochádzalo k zväčšeniu. Avšak nechceme, aby sa menila maska štetca, pretože tá definuje počet štetín a je pre jeden ťah nemenná. Nechceme meniť počet štetín v maske. Zmenu veľkosti stopy, ktorú štetec zanecháva, dosahujeme tak, že súradnice štetín transformujeme pomocou matice pre zmenu mierky. Tým vzniknú väčšie medzery medzi štetinami a zväčší sa tvar. Pri veľkých hodnotách rozpoznávame jednotlivé štetiny. Tento efekt však umelci dokážu využiť ako vidíme napr. na obrázku 35. Samy Lang použil Hairy brush na tvorbu dažďa v maľbe pomenovanej *Rain*. Zmena mierky je namapovaná na tlak, ktorý poskytuje tablet. Väčším tlakom sa štetiny od seba počas ťahu štetca viac oddeľujú, dochádza k rozprestreniu ich pozícií.

Ďalší parameter je skosenie. Užívateľ definuje skosenie a tým mení tvar štetca. Tento parameter je inšpirovaný skosenými tvarmi štetcov, ktorých použitie je časté s kaligrafickými ťahmi. Jednotlivá štetina je teda transformovaná náhodným vektorom posunutia, zmenou mierky a následne skosením.

Po transformácií vykresľujeme trajektóriu štetiny. Spočítame počiatočnú a koncovú súradnicu štetiny tak, že ju posunieme do počiatočného a koncového bodu úsečky ťahu a pomocou algoritmu pre rasterizáciu úsečky spočítame vektor polôh pixlu. Môžeme použiť algoritmus Bresenshama alebo DDA algoritmus. Pri výpočte dráhy štetiny si musíme pamätať predchádzajúcu transformovanú polohu štetiny, pretože chceme aby dráha štetiny bola spojitá. Táto možnosť je v štetci však voliteľný parameter. Môžeme totiž vygenerovať novú počiatočnú súradnicu. Prítomnosťou náhodného posunutia bude vždy iná. Tým sme umožnili vytvoriť „dažďový efekt“ v spomínanej maľbe *Rain*.

Namiesto vykresľovania pozícií si ukladáme do vektora jednotlivé pozície pixlov rasterizovanej dráhy. Algoritmy pre rasterizáciu väčšinou pracujú s celočíselnými súradnicami. Počiatočné a koncové súradnice sú však reálne. Sub-pixlová presnosť je teda zanedbaná. Problém môžeme riešiť napríklad tak,že upravíme algoritmus rasterizácie tak, aby pracoval s reálnymi číslami. Prípadne po spočítaní dráhy pixlov lineárne interpolu-

jeme informáciu o sub-pixel pozícii a pripočítame ju k pozícii každého pixlu. Informáciu o sub-pixel pozíciách môžeme využiť pre anti-aliasing dráhy štetiny.



Obrázok 20: Wu častica: vľavo častice s reálnymi súradnicami, vpravo ich vykreslenie

Anti-aliasing dráhy štetiny môžeme riešiť pomocou častíc označovaných ako Wu pixel [2]. Štetiny reprezentujeme pri vykresľovaní pixlom. Súradnice pixlu sú reálne. Práve tento fakt využíva Wu pixel, pre ktorý platí, že celkový jas vykreslených pixlov sa rovná jasu častice. Jeden pixel je vykresľovaný maximálne štyrmi pixlami, ktorých súčet jasov je rovný pôvodnému jasu častice ako vidíme na obrázku 20. Pri vykresľovaní počítame oblasť, ktorú pixel pokrýva. Namiesto výpočtu intenzity, ktorú popisuje Hugo Elias [2], môžeme počítať hodnotu alfa kanálu.

```
int alpha = color.alpha();
int ipx = int (pixelPosition.x());
int ipy = int (pixelPosition.y());
float fx = pixelPosition.x() - ipx;
float fy = pixelPosition.y() - ipy;

int btl = round((1.0 - fx) * (1.0 - fy) * opacity);
int btr = round((fx) * (1.0 - fy) * opacity);
int bbl = round((1.0 - fx) * (fy) * opacity);
int bbr = round((fx) * (fy) * opacity);

color.setOpacity(btl); setPixel(ipx , ipy, color);
color.setOpacity(btr); setPixel(ipx + 1, ipy, color);
color.setOpacity(bbl); setPixel(ipx , ipy + 1, color);
color.setOpacity(bbr); setPixel(ipx + 1 , ipy + 1, color);
```

Výpis 7: Výpočet oblasti a vykreslenie Wu pixlu

Pozície štetiny sme si ukladali do vektoru preto, aby sme mohli vytvárať farebné efekty. Efekt, ktorý chceme simulovať je efekt vyprázdňovania atramentu v štetci. Každá štetina obsahuje na začiatku 100% atramentu a má určenú svoju dĺžku. Práve tieto atribúty štetiny používame pri výpočte farby vykresľovanej farby štetiny. Do úvahy be-

rieme navyše aj tlak, ktorý poskytuje vstupné zariadenie. Užívateľ má ďalšiu možnosť definovať priebeh vyprázdňovania štetca pomocou krivky.

Obrázok 21: Hairy brush: simulácia míňania atramentu štetca

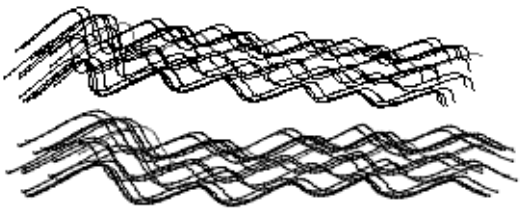
Najlepšie simuluje vyprázdňovanie atramentu zmena alfa kanálu farby. Postupné zmenšovanie hodnoty alfa kanálu vytvára efekt, ktorý pripomína míňanie farby. Postupné znižovanie realizujeme pomocou krivky, ktorú definuje užívateľ. Užívateľ navyše definuje celkové množstvo atramentu pre štetec. Hodnotu množstva atramentu použijeme pre vzorkovanie užívateľsky definovanej krivky. Každá štetina obsahuje počítadlo, ktoré je zvýšené pri každom kontakte s plátnom. Počítadlo slúži ako index do vektoru hodnôt vzorkovanej krivky. Vektor obsahuje hodnoty, ktoré používame pri výpočte transformácie farby a zároveň na danú hodnotu nastavujeme množstvo atramentu v štetine. Krivka vyjadruje priebeh míňania sa atramentu počas ťahu. Množstvo atramentu v štetine je modelované pomocou priehľadnosti. Bolo testované i postupné míňanie saturácie farby. To sa ukázalo ako neužitočný efekt, ktorý vytvára artefakty.

```
for (int i = 0; i < bristleCount;i++){
   bristle = bristles[i];
   float alpha = bristle.length() * bristle.inkAmount();
   plotBristle ( bristle );
   bristle.setInkAmount( inkDepletionCurve[ bristle.counter() ] );
}
```

Výpis 8: Výpočet jednoduchého vyprázdňovania farby pomocou zmeny alfa kanálu

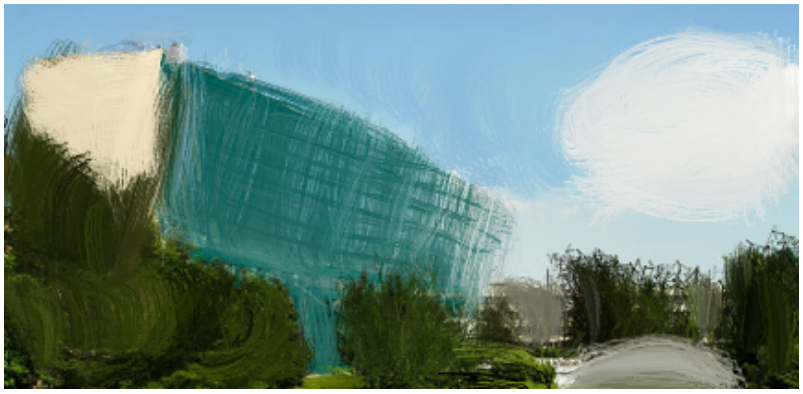
Štetinu vykresľujeme tak, že prechádzame vektorom pozícií, ktoré sme spočítali pomocou algoritmu na rasterizáciu úsečky.Rozmer štetiny je jeden pixel. Pri použití možnosti anti-aliasingu je pixel vykresľovaný ako Wu častica. Rozdiel kvality výstupu vidíme na obrázku 22. Pre každú pozíciu spočítame farbu a vykreslíme pixel do pomocného pixel bufferu. Je dôležité zvoliť správnu kompozitnú operáciu. Pokiaľ by sme kopírovali farbu štetiny do pomocného pixel bufferu, tak prepisujeme farby štetín, ktoré prechádzali daným miestom. Pokiaľ má štetec jednu farbu, stačí ak použijeme kompozitný režim, pri ktorom zohľadňujeme alfa kanál farby. Porovnáme hodnoty alfa kanálu dvoch pixlov a pokiaľ má nový pixel hodnotu alfa kanálu väčšiu, prepíšeme hodnotu alfa kanálu starého pixlu novou hodnotou. Tento kompozitný režim sa podobá kompozitnému režimu alfa-stmavenie (anglicky *alpha-darken*). Tiež dochádza k postupnému stmavovaniu pixlov.

Pokiaľ je maska štetca farebná, môžeme použiť radšej alfa-miešanie, pretože inak by sme ignorovali farby štetín.



Obrázok 22: Hairy brush: porovnanie stopy štetca s aliasom a bez aliasu

Ďalší farebný efekt je jednoduchá simulácia obojsmerného prenosu farby medzi štetcom a plátnom. Simulácia tohto efektu tak, aby odpovedala skutočnosti, je náročný problém. Fyzikálny model popisuje vo svojej práci Bill Baxter[1]. Model sme obmedzili iba na to, že pri prvom kontakte štetca s plátnom sú všetky štetiny zafarbené farbou plátna v mieste styku štetiny s plátnom. Tento efekt má za cieľ opäť riešiť problém *„put-that-color-here“*, ktorý sme spomínali v časti 2.1. Keďže štetina má rozmer jeden pixel, farba je vzatá z daného pixlu plátna. Pozícia štetiny posunutá do pozície v plátne je reálna. To znamená, že môžeme výslednú farbu interpolovať z okolitých pixlov pomocou lineárnej interpolácie alebo môžeme jednoducho vziať farbu pixlu metódou najbližšieho suseda. Tento efekt môžeme využiť na premenu fotografie na umelecké dielo. Obrázok 23 je demonštráciou tejto vlastnosti štetca. Efekt je inšpirovaný taktiež prácou Paula Haeberliho [3], efekt je možné vidieť v jeho programe *The Impressionist* [6].



Obrázok 23: Hairy brush: obojsmerný prenos farby z plátna na štetec

Použitie štetca Hairy brush je vo workflow umelca pri tvorbe textúry objektov, pri kresbe vlasov, trávy, srsti a podobne. Ďalej je tento typ štetca použitý pri maľbe sumi-

[6] Java applet dostupný na `http://laminadesign.com/explore/impression/index.html`

e štýlu, čo sú maľby vytvárané čínskym štetcom. Tento štetec je odlišný od klasických štetcov, ktoré pečiatkujú masku štetca podľa definovanej medzere. Umožňuje vytvárať originálne stopy ťahov štetca, ktoré sa nedajú reprodukovať pomocou pečiatkových štetcov, prípadne veľmi ťažko a neefektívne.

3.3 Spray brush

Ďalším štetcom, ktorý bol implementovaný v prostredí aplikácie Krita je sprej (anglicky Spray brush). Ako jeho názov naznačuje, úlohou je simulácia spreja alebo rozstrekovača. Motiváciou pre jeho realizáciu je zefektívniť vytváranie textúr objektov v maľbe a simulácia vlastností skutočného nástroja s názvom Airbrush. Obdobný štetec je prítomný v grafických softvérových nástrojoch pod názvom *Airbrush*, ktorý simuluje rovnomenné skutočné zariadenie. Airbrush je zariadenie, ktoré rozstrekuje farbu alebo atrament pomocou natlakovaného vzduchu veľkou rýchlosťou. Používa sa na maľovanie umeleckých diel technikou akvarelu (vodovej farby) alebo na nástenné maľby. Skúsený umelec pomocou Airbrushu dokáže vytvárať obrazy s fotorealistickou kvalitou a prakticky simulovať ľubovoľnú umeleckú techniku. V minulosti sa používal na úpravu fotografií, dnes sa používajú editory rastrovej grafiky. Jeho použitie siaha od rôznej formy umeleckej činnosti ako je napríklad pouličné umenie až po použitie v automobilovom priemysle na auto-motív [13]. Softvérová verzia slúži na koncept art alebo na tvorbu návrhu.

Typická softvérová implementácia je v grafických nástrojoch obmedzená na opakovanú kompozíciu masky štetca s plátnom, pričom sa dynamicky mení priehľadnosť masky prípadne jej mäkkosť. Parametrom zmeny priehľadnosti najčastejšie býva tlak tabletu. Užívateľ kontroluje aj rýchlosť kompozície masky štetca s plátnom. Maska štetca je pritom statická, prípadne sa mení jej veľkosť. Túto funkciu dokáže simulovať štetec Soft brush. V spreji sme sa pokúsili o iný model tohto štetca. Opäť používame častice na modelovanie spreja. Jednotlivé častice s rôznymi atribútmi ako je veľkosť, rotácia alebo farba dopadajú na plátno, pričom ich stav sa už ďalej nemení. Generujeme dynamickú masku štetca tvorenú časticami. Ťah štetca môžeme prirovnať k počítačovej animácii. Štetec generuje jednotlivé rámce animácie, ktoré sú následne spojené s obrazom plátna na pozíciu vstupného zariadenia v plátne. Jednotlivé rámce sú navzájom nezávislé, pre každý rámec je vygenerovaná nová maska štetca.

Obrázok 24: Spray brush: textúra trávy vytvorená štetcom

V prvom kroku vymedzíme oblasť, do ktorej jednotlivé častice spreja dopadajú. Tento tvar môže byť definovaný užívateľom pomocou procedurálnej masky štetca alebo môžeme použiť digitálny obraz ako vstup. Pri použití procedurálnej masky budeme

vytvárať eliptický tvar. Ten je typický pri reálnych štetcoch aj pri Airbrush štetci. Eliptický tvar je zároveň dobre kontrolovateľný pomocou dynamických parametrov, ktoré poskytuje tablet. Sklon tabletu môžeme použiť na zmenu pomeru strán. Rotáciou tabletu môžeme rotovať eliptickým tvar a tlak tabletu mapujeme na veľkosť eliptickej oblasti. Užívateľ definuje vlastnosti eliptickej oblasti ako je priemer, pomer strán, zmenu mierky a rotáciu. Na základe týchto vlastností generujeme polárne súradnice častíc. Polárne súradnice generujeme preto, aby sme generovali súradnice v eliptickej oblasti efektívne.

Jednoduchý štetec, ktorý simuluje sprej pixlov v programe typu Maľovanie používa nasledovný kód[7] na generovanie pixlov v kruhu.

```
int radius = 50;
...
for (int i = 0 ; i < particleCount; i++){
  float rx = random(0,1) * radius;
  float ry = random(0,1) * radius;

  if (radius*radius <= rx*rx + ry*ry)
    addPoint(rx,ry);
}
```

Výpis 9: Generovanie pixlov v kruhu

Tento prístup má niekoľko nedostatkov a nevýhod v spojení s generovaním budov v eliptickej oblasti. Funkcia *random()* vracia pseudo-náhodné reálne číslo v intervale (0,1) s distribúciou, ktorá je uniformná. Generuje každú hodnotu s rovnakou pravdepodobnosťou. To znamená, že veľa hodnôt je neplatných pri generovaní súradníc, pretože sú generované v štvorcovej alebo obdĺžnikovej oblasti a padajú mimo požadovaný kruh. Navyše tento kód generuje súradnice len v kruhu a my požadujeme eliptický tvar. Kvôli rýchlosti odozvy často chceme kontrolovať počet častíc, ktoré štetec doručí na plátno počas generácie jednej masky. Užívateľ definuje počet častíc konkrétnou hodnotou alebo percentuálne vyjadrí pokrytie štetca časticami. Kvôli neplatným súradniciam, ktoré sú v tomto prípade generované, by sme túto požiadavku spĺňali neefektívne, prípadne vôbec nespĺňali. Mohli by sme neefektívne generovať nové častice pre každú neplatnú časticu.

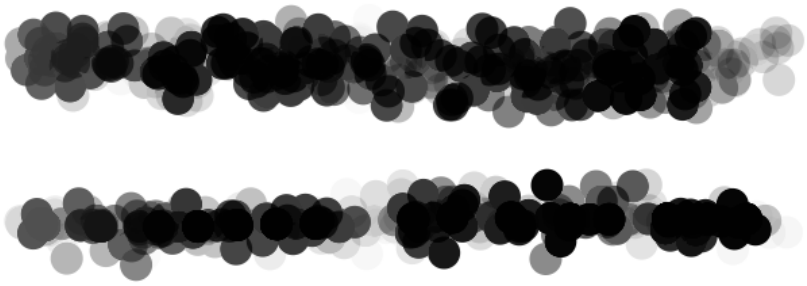
Problém riešime použitím polárnych súradníc. Tieto súradnice definujú bod v priestore pomocou vzdialenosti r od počiatku súradnej sústavy a pomocou uhla θ. Jednoducho môžeme generovať body na kružnici s daným konštantným polomerom. Pokiaľ chceme generovať body v kruhu, r obmedzíme na interval. Prípadne môžeme generovať body v kruhovom výseku, v tomto prípade obmedzíme aj uhol θ na interval.

$$x = r * \cos(\theta) \tag{3}$$

$$y = r * \sin(\theta) \tag{4}$$

[7] Obdobný kód sa nachádza v open-source implementácií Maľovania v prostredí KDE s názvom KolourPaint

Polárne súradnice môžeme jednoducho previesť na kartézske súradnice pomocou vzťahov 3 a 4. Keďže však chceme generovať body v eliptickej oblasti, transformujeme po prevedení súradnicu y parametrom pomer strán obdĺžnika ohraničujúceho tvar elipsy. Pre sprej generujeme častice, ktoré majú polohu v 2D priestore. Pre každý bod generujeme uhol θ, vzdialenosť r. Uhol θ generujeme pomocou funkcie s rovnomerným rozdelením pravdepodobnosti. V spreji chceme aby častice rovnomerne pokrývali oblasť vzhľadom na uhol. Vzdialenosť r generujeme buď rovnomerným rozdelením pravdepodobnosti alebo užívateľ môže zvoliť normálne rozdelenie pravdepodobnosti resp. Gaussovo rozdelenie pravdepodobnosti. Chceme tým dosiahnuť to, že častice spreja sa nám viac zhlukujú okolo stredu súradnej sústavy. Väčšina častíc je totiž bližšie stredu a len niekoľko sa ich nachádza po okraji. Rozdiel medzi uniformnou a Gaussovou distribúciou vidíme na obrázku 25.



Obrázok 25: Distribúcia častíc v štetci: uniformná a Gaussova distribúciou

Pri určení oblasti, do ktorej budú častice dopadať, môžeme použiť digitálny obraz. Postup sa líši od generovania častíc procedurálne pre eliptickú oblasť. Digitálny obraz môže každým pixlom určovať platnú polohu častice. Určíme stred súradnicovej sústavy v obraze a následne spočítame polohu pixlu v tejto súradnicovej sústave. Stred súradnicovej sústavy môže určiť užívateľ alebo ho automaticky položíme do stredu vstupného digitálneho obrazu. Z obrazu vytvoríme vektor polôh pre častice. Do vektoru polôh zaradíme len niektoré pixle, tak aby sme oproti generovaniu eliptickej oblasti získali možnosť generovať komplikovanejšie tvary. Hodnotiacim kritériom na zaradenie alebo nezaradenie daného pixlu môže byť hodnotu alfa kanálu alebo intenzita farby prípadne obe vlastnosti. Transparentné pixle prípadne biele, jasné pixle nezaradíme do výsledného vektora častíc. Pre každú časticu zároveň potrebujeme spočítať uhol, ktorý zviera úsečka daná počiatkom súradnicovej sústavy a polohou častice s osou x. Tento uhol používame pre efekty rotácie častíc. Intenzita farby pixlu môže určovať ďalšie vlastnosti častice ako je napríklad konštantná rotácia častice alebo určitá farebná vlasnosť.

Ďalšou požiadavkou štetca je, aby ťahy štetca boli rôznorodé. Napríklad pomocou tohto štetca bola vytváraná textúra trávnatého povrchu na obrázku 24. Postačí nám jedna textúra stebla trávy, ktorú následne rotujeme, zmenšujeme a zväčšujeme pri generovaní masky štetca. Častica okrem polohy bude charakterizovaná svojou veľkosťou,

respektíve mierkou. Užívateľ môže určiť konštantnú veľkosť častice prípadne môže definovať relatívnu veľkosť častice.V prípade konštantnej veľkosti sú všetky častice rovnaké. Užívateľ môže definovať veľkosť častice priamo hodnotou v pixloch alebo relatívne vzhľadom na veľkosť oblasti štetca v percentách. Relatívna veľkosť je dôležitá pre workflow, ktorý používa umelec pri maľovaní. Umožňuje mu meniť veľkosť plochy, na ktorú častice dopadajú, pričom častice menia veľkosť relatívne k ploche a teda nemusí ich nastavovať. Mierku častice môže ovplyvniť ďalším voliteľným parametrom, a to náhodnou veľkosťou. Náhodnú veľkosť častíc vidíme na obrázku 26. Vygenerované náhodné číslo v intervale (0,1) je použité na zmenu mierky častice. Zaujímavý efekt dosahuje veľkosť častíc pri použití funkcie sin alebo cos s inkrementovaným počítadlom pre každú vygenerovanú masku. Absolútna hodnota funkčných hodnôt sa pohybuje v intervale (0,1). Častice sa pritom postupne zväčšujú a zmenšujú. Prípadne môžeme užívateľovi poskytnúť možnosť definovať priebeh zmeny veľkosti pomocou krivky.

Obrázok 26: Veľkost častíc: náhodná veľkosť častíc v ťahu

Pre každú časticu určíme jej rotáciu. Uhol rotácie pre časticu je len jeden. Pri vykresľovaní rotujeme časticu okolo vlastnej osi. Zaujímavým rozšírením môže byť rotácia častice v 3D priestore. V implementácii používame viaceré spôsoby nastavenia rotácie častice. Užívateľ môže definovať konštantný uhol rotácie pre všetky častice. Ďalej náhodný uhol, ktorý môže sledovať distribúciu s normálnym rozdelením alebo rovnomerným rozdelením pravdepodobnosti. Ďalšia možnosť je použitie uhla, ktorý bol vygenerovaný pri procese generácie polárnych súradníc. Pokiaľ použijeme uhol θ ako uhol rotácie, pre užívateľa častice sledujú polohu kurzora vstupného zariadenia v plátne. Na obrázku 27 vidíme konštantný uhol rotácie častíc, náhodný uhol rotácie a možnosť, kedy používame uhol θ. Ďalšia možnosť rotácie je sledovanie ťahu štetca. Uhol spočítame podobne ako pre štetec Hairy brush v časti 3.2. Efekt demonštruje obrázok 28.

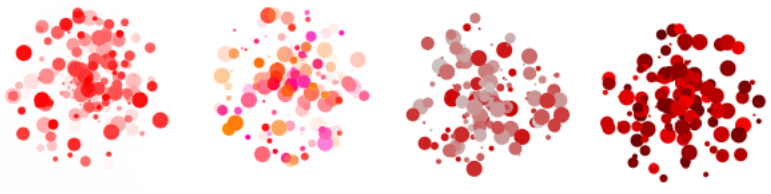
Obrázok 27: Rotácia častíc v štetci: konštantná, náhodná a uhol θ

Užívateľ môže použiť ľubovoľnú kombináciu možností. Navyše sme pridali možnosť určiť váhu jednotlivým možnostiam rotácie. To znamená, že jednotlivé uhly rotácie môžu byť navzájom závislé, a tým dosahujeme ďalšie zaujímavé efekty. Váhy užívateľ môže nastavovať pre náhodný uhol a pre uhol θ. Pokiaľ užívateľ aktivuje všetky možnosti zadania uhla rotácie častice, najprv dochádza k lineárnej interpolácií medzi konštantnou hodnotou rotácie a hodnotou náhodnej rotácie, následne je výsledok lineárne interpolovaný s hodnotou uhla θ. Jednotlivé váhy môžeme mapovať na dynamické atribúty tabletu a tým užívateľ môže viac kontrolovať výslednú rotáciu častíc.

Obrázok 28: Rotácia častíc v štetci: sledovanie ťahu štetca

Pomocou štetca chceme simulovať maľbu akvarelovými farbami. Tieto maľby sú charakteristické širokou škálou farebných odtieňov. V digitálnej maľbe je problém vytvárať takéto maľby, pretože užívateľ musí vyberať farby z palety a táto operácia je pomalá. V tomto štetci sa pokúsime riešiť tento problém tak, že každá vykreslená častica bude mať vlastnú farbu. Prvý parameter, ktorý ovplyvňuje farbu častice je náhodná priehľadnosť. Pri zvolení tejto vlastnosti má častica náhodnú priehľadnosť vzhľadom na zvolenú farbu. Priehľadnosť častice môže zaujímavo ovplyvňovať jej poloha, respektíve vzdialenosť od počiatku súradnicovej sústavy. Taktiež môžeme vytvoriť závislosť medzi veľkosťou častice a jej priehľadnosťou.

Ďalšia premenlivá vlasnosť farby je dynamická zmena vo farebnom priestore HSV. Je podobná ako pri štetci Soft brush v časti 3.1. Vychádza z farebného modelu HSV. Užívateľ definuje veľkosť zmeny jednotlivého kanálu modelu HSV. Vo farebnom modeli HSV je zložka odtieň (hue) vyjadrená uhlom v rozsahu 0 až 360 stupňov. Užívateľ môže definovať zmenu v rozsahu 0 až 180 stupňov v oboch smeroch rotácie. Sýtosť (saturation) a intenzita (value) je definovaná v rozsahu 0 až 100%. Užívateľ vyberá veľkosť zmeny v tomto rozsahu. Náhodne generované číslo je namapované na odpovedajúce si intervaly a farba štetca je transformovaná pomocou náhodných hodnôt vo zvolených intervaloch. Generovanie náhodnej priehľadnosti a vlastností HSV modelu môže prebiehať v rámci jednej vygenerovanej masky pre každú časticu samostatne alebo náhodné hodnoty môžu častice v rámci jednej masky zdieľať.



Obrázok 29: Spray brush: farby a náhodná priehľadnosť, náhodný odtieň, náhodná sýtosť a náhodný jas

Zdrojom farby pre časticu môže byť i plátno podobne ako v štetci Hairy brush, kde štetina mohla pri prvom kontakte s plátnom získať farbu. V štetci Spray brush je taktiež tento efekt prítomný. Po vygenerovaní polohy častice v rámci masky je častica posunutá o polohu vstupného zariadenia v plátne a vykreslená s farbou pixlu plátna v kontaktnom mieste. Týmto efektom simulujeme umelecký smer impresionizmus. Z fotografie vytvoríme rýchlo a jednoducho umelecký obraz. Efekt demonštrujeme na obrázku 30. Obdobný postup používa program Paula Haeberliho *The Impressionist*. Farba vzatá z plátna je pred vykreslením spracovaná spomínanou HSV transformáciou a môže mať náhodnú priehľadnosť.

Obrázok 30: Spray brush: impresionizmus pomocou štetca z fotografie

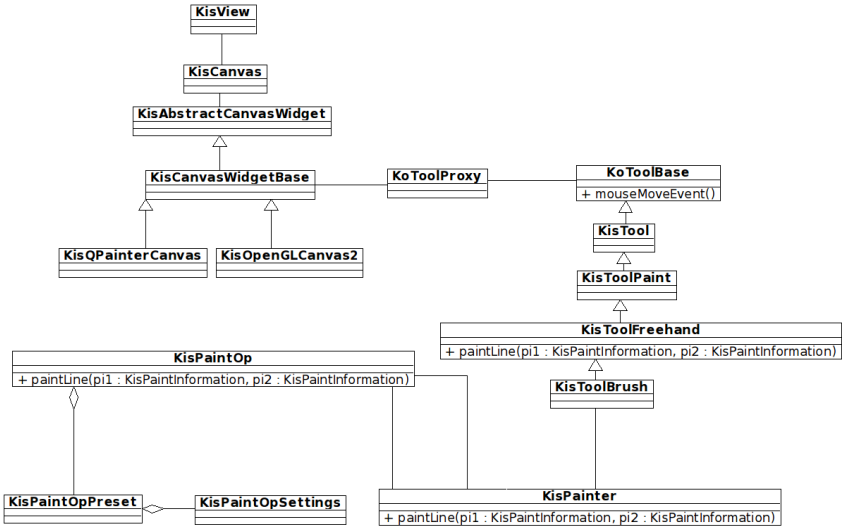
Samotná častica má taktiež svoj tvar. Môže to byť procedurálna maska, ľubovoľný tvar ako je elipsa alebo obdĺžnik alebo textúra. V rámci implementácie podporujeme i častice veľkosti pixlu a Wu častice. Spray brush dokáže teda simulovať v rôznych nastaveniach komplikované štetce ale takisto i obyčajný sprej z aplikácie typu Maľovanie.

4 Analýza a testovanie

4.1 Integrácia do programu Krita

Všetky navrhnuté štetce boli integrované a implementované v programe Krita. Program Krita je open-source projekt. Ide o KDE program na maľovanie a tvorbu skíc. Cieľom tohto programu je poskytnúť umelcom, ktorí produkujú umenie v digitálnej forme, kompletné riešenie pre digitálne maľovanie. Krita explicitne podporuje workflow pre koncept art, tvorbu komixov a textúr, ktoré môžu byť použité pri renderingu.

Krita je implementovaná v jazyku C++ použitím rôznych knižníc z prostredia KDE,frameworku Qt a iných. Architektúra Krity je značne postavená na zásuvných moduloch. Obrazový filter, farebný priestor, nástroje a štetce sú zásuvné moduly. Na obrázku 31 vidíme triedy, ktoré sú dôležité z pohľadu implementácie nového štetca.



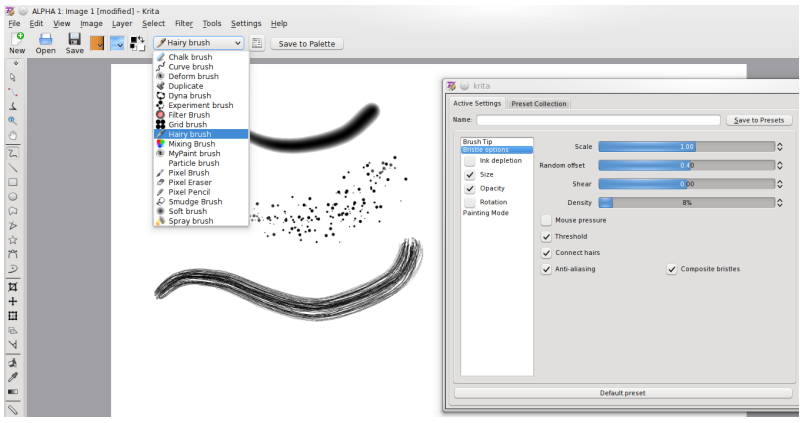
Obrázok 31: UML diagram tried: od plátna k štetcu

Trieda *KisView* sa stará o zobrazenie dokumentu. Pozostáva mimo iných tried i z triedy *KisCanvas*, ktorá reprezentuje plátno. Plátno môže byť vykreslované pomocou triedy QWidget alebo QGLWidget z frameworku Qt. QWidget je vykresľovaný pomocou softvérového renderingu na platforme Windows, pomocou serveru X11 na platforme Linux a tak ďalej. QGLWidget je vykresľovaný pomocou OpenGL. Viac informácií je dostupných v dokumentácii Qt frameworku [8]. *KisCanvas* je teda triedou, ktorá plní úlohu adaptéru. Pozostáva z triedy *KisAbstractCanvasWidget*, ktorá tvorí abstraktný interface. Tento interface implementuje trieda *KisCanvasWidgetBase*, z názvu vidíme, že ide o základnú triedu, ktorú implementujú oba typy tried použitých na vykresľovanie

[8] Vykresľovací systém Arthur `http://qt.nokia.com/doc/4.6/qt4-arthur.html`

plátna. Obsahuje i metódy, ktoré mapujú pozície udalosti vstupného zariadenia v plátne na dokument. Posun myši z pozície v plátne na základe zvolenej miery zoomu pretransformuje na pozíciu v dokumente, ktorým je digitálny obraz. *KisCanvasWidgetBase* používa triedu *KoToolProxy* ako zástupcu všetkých možných nástrojov, cez túto triedu posiela udalosti nástrojom.

Nástrojov v Krite je viac, napríklad nástroj kresliaci elipsu pomocou zvoleného štetca. Najdôležitejší nástroje je však nástroj kresliaci voľnou rukou. Umožňuje vytvárať ľubovoľné ťahy štetca pohybom myši alebo tabletu. *KoToolProxy* používa triedu *KoToolBase*, ktorá obsahuje i metódu mouseMoveEvent(), ktorá je volaná vtedy, keď sa myš či tablet hýbe po plátne. Následná hierarchia tried slúži na rozlíšenie rôznych typov nástrojov. Nástroj kresliaci voľnou rukou patrí do množiny kresliacich nástrojov. Implementáciu tohto nástroja obsahuje trieda *KisToolFreehand*. V implementácií vytvára z pohybu myši alebo tabletu volania paintLine() s parametrami pi1 a pi2 triedy *KisPaintInformation*. Tieto parametre nesú informácie ako je pozícia štetca v dokumente, tlak, sklon a rotácia tabletu, vektor posunu a jeho uhol a iné. Volania paintLine() posiela na dôležitú triedu *KisPainter* a sú kvôli presnosti ukladané do fronty, ktorá je realizovaná pomocou separátneho vlákna. Jedno volanie paintLine() môže zabrať čas procesoru, a ten by nemohol prijímať udalosti pohybu vstupného zariadenia od operačného systému. Trieda *KisToolBrush* je špecializáciou triedy *KisToolFreehand*. Dopĺňa užívateľské rozhranie a zároveň zlepšuje funkčnosť štetcov. Pokiaľ nehýbeme vstupným zariadením, negenerujeme nové udalosti a nedochádza k novým volaniam paintLine(). Táto trieda obsahuje časovač, ktorého frekvenciu môže užívateľ konfigurovať. V časových odstupoch generuje volania paintAt() triedy *KisPaintOp*. Táto funkčnosť je dôležitá pre podporu štetcov, ktoré pracujú podobne ako zariadenie Airbrush simulované štetcami Soft brush alebo Spray brush.



Obrázok 32: Krita: integrované štetce v ponuke štetcov a konfiguračný dialóg

Triedu *KisPainter* je určená na vykresľovanie do triedy *KisPaintDevice*, ktorá nesie digitálny obraz plátna, je to vlastne pixel buffer. Ide o zložitejší pixel buffer, ktorý pod-

poruje rôzne farebné priestory (RGB, CMYK) s rôznou bitovou hĺbkou na kanál (8-bit, 16-bitová hĺbka). Celý pixel buffer je reprezentovaný dlaždicami a nie je to lineárny kus pamäti určitých rozmerov. Dlaždice sú mále lineárne oblasti pamäte, ktoré tvoria celý výsledný pixel buffer. Táto reprezentácia je dôležitá kvôli možnosti návratu stavu pixel bufferu pomocou funkcie späť (anglicky *undo*), ale i kvôli výkonu. Navyše tento pixel buffer môže byť neobmedzene veľký, respektíve obmedzuje ho len veľkosť dostupnej pamäti. Krita podporuje vrstvy, takže každá vrstva má svoj pixel buffer. Dôležitá funkcia z pohľadu štetcov je funkcia blokového prenosu bitBlt(), ktorá kompozitnou operáciou spája pixel buffre do výsledného obrazu. Užitočné pre štetce sú i rutiny na rasterizáciu objektov ako je priamka, elipsa a podobne. Spray brush používa rutinu na rasterizáciu polygónu.

KisPainter pozostáva z triedy *KisPaintOp*, ktorá je používaná kresliacimi nástrojmi (PaintOp je skratka z anglicky Painting operation). Všetky realizované štetce implementujú túto triedu. *KisPaintOp* obsahuje volanie paintLine(), ktoré je volané v rovnomennej funkcii triedy *KisPainter*. Trieda *KisPaintOp* obsahuje už aj implementáciu tejto metódy, konkrétne vzorkuje úsečku ťahu na jednotlivé pozície podľa definovanej medzere štetca (anglicky *spacing*). Pre jednotlivé pozície ťahu vytvára volanie paintAt() s parametrom info typu *KisPaintInformation*.

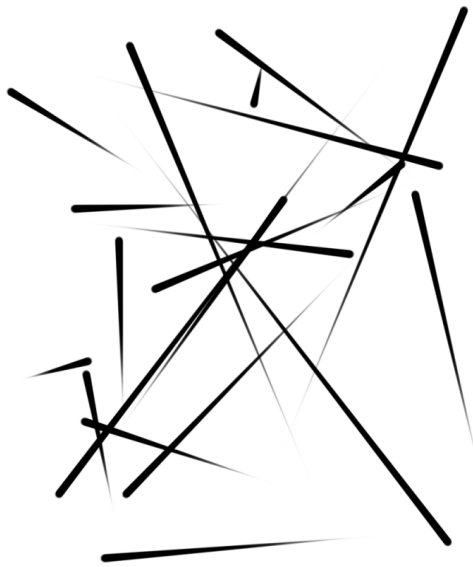
Pri implementácií nového štetca dedíme z tejto triedy, musíme poskytnúť implementáciu metódy paintAt(), ktorá je virtuálna v triede *KisPaintOp*. Tento postup používajú implementované štetce Soft brush a Spray brush. Ďalšia možnosť je preťaženie metódy paintLine(). Túto možnosť využívame v štetci Hairy brush. *KisPaintOp* obsahuje odkaz na *KisPainter*, pomocou ktorého môžeme kompozitnou operáciou spojiť vlastný vytvorený pixel buffer s pixel buffrom plátna na odpovedajúce pozíciu, ktorú poskytujú parametre funkcií paintAt() alebo paintLine().

Každý štetec poskytuje širokú škálu efektov. Rôzne efekty sú dosiahnuté pomocou konfigurácie stavu štetca. Užívateľ konfiguruje každý štetec zvlášť. Niektoré voľby sa opakujú ako je napríklad medzera, maska štetca a podobne. Pre každý štetec bolo implementované vlastné užívateľské rozhranie. Stav užívateľského rozhrania je serializovaný a deserializovaný pomocou triedy *KisPaintOpSettings*. Tá je súčasťou triedy *KisPaintOpPreset*, ktorá umožňuje jednotlivé serializované stavy štetca ukladať na disk. Súborový formát je PNG obrázok a v textovej oblasti tohto formátu je uložený stav užívateľského rozhrania v XML. Užívateľ si môže konfigurácie ukladať a neskôr ich znovu použiť. Podpora pre túto funkčnosť bola pridaná pre všetky implementované štetce.

4.2 Testovanie

Každý štetec bol optimalizovaný počas niekoľkých iterácií. V tejto časti uvádzame finálne verzie a ich profiláciu pomocou nástroja *Valgrind*. Zároveň boli vytvorené testy použitím QTestLib frameworku, ktorý je súčasťou Qt 4.6. Vytvorili sme merania pre všetky štetce a testovali sme viacero konfigurácií. Každý štetec v testoch vykresľuje jeden ťah s meniacim

sa tlakom, na čo štetec zvyčajne reaguje zmenou veľkosti masky. Druhý test vykresľuje 20 priamok s počiatočným tlakom na minimálnej hodnote 0.0 a s koncovým tlakom na maximálnej hodnote 1.0. Výstup testu vidíme na obrázku 33 Pri meraní času oba testy bežali 10-krát. Použitý počítač na testovanie má procesor Intel Core2 Duo CPU P7350, frekvencia 2.00 GHz, veľkosť pamäte RAM 2 GB, grafická karta NVidia 9200M. Použitý operačný systém Linux, distribúcia Fedora 12, 32-bitová verzia, kernel 2.6.32.9. Kompilátor gcc vo verzií 4.4.3. Pri testoch uvádzame čas vykreslenia a funkcie, ktoré sú relatívne pri vykresľovaní štetca a relatívny počet inštrukcií presahuje 10%. Veľkosť testovaného plátna 4096x4096 pixlov.



Obrázok 33: Výstup testu s 20 priamkami s premenlivým tlakom

Soft brush

Štetec Soft brush sme testovali s dvoma konfiguráciami. V základnej konfigurácií sme zvolili 30 pixlov široký kruhový štetec vo východzej konfigurácií, kedy na tlak reaguje štetec zmenou veľkosti a zmenou priehľadnosti. Ostatné vlastnosti boli neaktívne.

Test	Čas (v milisekundách)
Beziérová krivka	301.8
Náhodné čiary	1,618.2

Tabuľka 1: Rýchlosť štetca Soft brush

Z výsledkov vidíme, že štetec poskytuje veľmi rýchlu odozvu. Testovaním v aplikácií sme zistili, že odozva štetca je dobrá i pre veľmi veľké masky v rozmeroch 800

pixlov. V tabuľke 2 vidíme, že takmer 48% času strávi procesor vo funkcií paintAt(), v ktorej dochádza k rasterizácií masky štetca a jej vykresleniu. Na výpočet mäkkosti masky je použité zanedbateľné množstvo inštrukcií, mäkkosť masky je vopred počítaná z užívateľom definovanej krivky. Vo výpočte masky najviac času trávi procesor s kopírovaním pamäti pri opätovnom použití pixel bufferu. Alokovanie pamäti je pomalé, preto sa snažíme znovu použiť práve vykreslený pixel buffer, ktorý ale musíme vyprázdniť, pretože budeme generovať novú masku. Ďalej procesor trávi veľa času nastavovaním priehľadnosti pixlov funkciou KoColorSpaceAbstract::setOpacity(). Kruhové masky na prvý pohľad vyzerajú symetricky podľa horizontálnej aj vertikálnej osi. Mohli by sme teda spočítať iba štvrtinu masky a zvyšok pixlov skopírovať symetricky do pixel bufferu. Bráni nám však fakt, že maska zohľadňuje precíznosť na sub-pixel pozície, čo znamená že v skutočnosti nie je symetrická, pretože priehľadnosť pixlov sa jemne líši. Najmä na pixloch nachádzajúcich sa na hranách masky.

Štetec bol optimalizovaný tak, aby používal ako pomocný pixel buffer namiesto dlaždicovej pamäte reprezentovanej triedou *KisPaintDevice* lineárnu pamäť triedy *KisFixedPaintDevice*. Trieda *KisPaintDevice* nesie zo sebou transakcie súvisiace s funkciou späť, čo znamená ďalšiu pamäť a je tým tento typ pixel bufferu pomalší. Preto je i použitá funkcia KisPainter::bltFixed, ktorá cez funkciu KoColorSpace::bitBlt kompozitnou operáciou spája pomocný pixel buffer triedy *KisFixedPaintDevice* s pixel bufferom aktuálnej vrstvy triedy *KisPaintDevice*.

Funkcia	Počet inštrukcií	Relatívny počet inštrukcií
KisSoftPaintOp::paintAt	694 719 112	47.89
KisCurveMask::mask	428 392 385	29.53
KisPainter::bltFixed	231 714 905	15.97
KoColorSpace::bitBlt	202 404 520	13.95

Tabuľka 2: Výsledky profilácie Soft brush

V druhej konfigurácií sme testovali rýchlosť so všetkými vlastnosťami. Opäť veľkosť 30 pixlov, bola použitá eliptická maska, rotovaná o 45 stupňov. Zmena mierky bola nastavená na 200%, Zahrnuté bolo trasenie, dynamická zmena farby cez farebný priestor HSV bola nastavená na automatický rast. Na tlak reagovala rotácia masky, priehľadnosť, mäkkosť i veľkosť.

Test	Čas (v milisekundách)
Beziérová krivka	532.8
Náhodné čiary	3,013.3

Tabuľka 3: Rýchlosť štetca Soft brush v plnej konfigurácií

Vidíme, že v plnej konfigurácií je štetec pomalší takmer dvojnásobne. Je stále dostatočne rýchly a poskytuje odozvu v reálnom čase. Oproti predchádzajúcej konfigurácií viac času trávi vo funkcii generovania masky. To je spôsobené funkciou drand48(), ktorá generuje reálne čísla s uniformnou distribúciou. Pre každý pixel masky je vygenerované náhodné číslo pri realizácií funkcie hustoty. Generovanie náhodných čísel je pomalá operácia.

Funkcia	Počet inštrukcií	Relatívny počet inštrukcií
KisSoftPaintOp::paintAt	971 529 739	56.29
KisCurveMask::mask	807 218 627	46.77
drand48	279 017 018	16.17
erand48_r	229 361 447	13.29
memcpy	196 128 221	11.36

Tabuľka 4: Výsledky profilácie plnej konfigurácie Soft brush

Hairy brush

Pre štetec Hairy brush sme navrhli 4 testy. Prvý test je východzie nastavenie, zmenená je veľkosť masky na 30 pixlov a dynamicky reaguje na zmenu tlaku iba veľkosť. Počet štetín je daný kruhovou maskou, pričom je zvolená hustota 100%. Štetiny sú vykresľované bez použitia anti-aliasingu, teda Wu častíc.

Test	Čas (v milisekundách)
Beziérová krivka	1,232
Náhodné čiary	5,620

Tabuľka 5: Rýchlosť štetca Hairy brush v základnej konfigurácií

Hairy brush trávi najviac procesorového času vo funkcii paintLine(), kde prebieha rasterizácia a vykresľovanie štetín. Funkcie *getLinearTrajectory*, *addPoint* a *QVector::realloc* súvisia s triedou *Trajectory*.Trieda *Trajectory* obsahuje algoritmus pre rasterizáciu úsečky. Implementované boli algoritmy DDA a Bresenhamov algoritmus. DDA algoritmus bol pre naše použitie rýchlejší, a preto je použitý vo finálnej implementácii. Algoritmus totiž nevykresľuje priamky do pixel bufferu, ale ukladá pozície, po ktorých sa pohybuje štetina z východzieho bodu do koncového bodu priamky. Pozície ukladáme do triedy *QVector* a z profilácie vidíme, že dochádza k jeho častej realokácií. Pre každú štetinu je počítaná vlastná dráha. Implementácia bola optimalizovaná tak, aby nedochádzalo k realokácií pamäti pre každú štetinu, pretože táto operácia je pomalá. K realokácií dochádza iba vtedy, ak je nová dráha dlhšia ako posledná najdlhšia dráha štetiny ťahu.

Funkcie moveTo() tried *KisRandomConstAccessor* a *KisTiledRandomAccessor* súvisia s vykresľovaním štetín do pixel bufferu. Soft brush používa lineárnu pamäť na vykresľo-

vanie masky do pomocného pixel bufferu, Hairy brush používa dlaždicovú pamäť triedy *KisPaintDevice*. Pri vykresľovaní štetín nepoznáme veľkosť pomocného pixel bufferu vopred, pretože dráha štetín je náhodná. Preto používame triedu *KisPaintDevice*, ktorá umožňuje zapisovať do pixel bufferu ľubovoľnej veľkosti efektívne. Trieda sama spravuje potrebnú pamäť, stará sa o alokáciu dlaždíc a nealokuje zbytočne veľkú pamäť. Na prechádzanie pamäti je použitý iterátor s náhodným prístupom *KisRandomConstAccessor*. Pri zadaní súradníc pixlu v pomocnom pixel bufferi iterátor nájde odpovedajúcu dlaždicu pamäti a umožní z nej zápis alebo čítanie. Vyhľadanie odpovedajúcej častice je náročný proces. Pri vykresľovaní štetín pristupuje k jednotlivým pixlom pixel bufferu preto, lebo vykresľujeme štetinu, ktorá má rozmer pixlu prípadne rozmer Wu častice.

Funkcia	Počet inštrukcií	Relatívny počet inštrukcií
HairyBrush::paintLine	3 061 407 701	76.69
Trajectory::getLinearTrajectory	882 994 364	22.10
Trajectory::addPoint	747 879 819	18.72
KisRandomConstAccessor::moveTo	649 223 668	16.25
QVector::realloc	538 110 866	13.47
KisTiledRandomAccessor::moveTo	520 600 800	13.03

Tabuľka 6: Výsledky profilácie základnej konfigurácie Hairy brush

V časti 3.2 o Hairy brush sme tvrdili, že parameter hustota, ktorý kontroluje počet štetín, zlepšuje výkonnosť a odozvu štetca. Zároveň výrazne nemení charakter stopy, ktorú štetec vytvára pri určitých konfiguráciách. Druhý test, ktorý sme vytvorili, meral práve rýchlosť štetca s východzou konfiguráciou so zníženou hustotou na 30%. Z tabuľky 5 vidíme, že štetec je pomalší ako Soft brush. Avšak znížením hustoty sa približujeme k jeho výkonu ako vidíme v tabuľke 7. Znížením počtu štetín sa odozva štetca zrýchlila takmer 3-krát. Profilácia konfigurácie so zníženou hustotou vyzerá veľmi podobne ako v tabuľke 6.

Test	Čas (v milisekundách)
Beziérová krivka	478
Náhodné čiary	2,314

Tabuľka 7: Rýchlosť štetca Hairy brush v základnej konfigurácií

Doposiaľ sme testovali konfigurácie štetca bez použitia anti-aliasingu. Výpočet anti-aliasingu v počítačovej grafike je všeobecne náročný problém z pohľadu výkonnosti. Otestovali sme konfiguráciu zhodnú s konfiguráciou v prvom teste, tentokrát však štetiny vykresľujeme pomocou Wu častíc, ktorými realizujeme anti-aliasing.

Z tabuľky 8 vidíme, že v konfigurácií s anti-aliasingom je štetec 3-krát pomalší. Je to očakávaný výsledok. Každú štetinu totiž vykresľujeme až 4 pixlami, pričom používame

Test	Čas (v milisekundách)
Beziérová krivka	3,651
Náhodné čiary	16,679

Tabuľka 8: Rýchlosť štetca Hairy brush s použitím anti-aliasingu

na prístup do pamäti rovnaký postup ako pri vykresľovaní bez anti-aliasingu. V tabuľke s údajmi o profilácii vidíme, že najviac času trávi procesor pristupovaním k pixlom pixel bufferu.

Funkcia	Počet inštrukcií	Relatívny počet inštrukcií
HairyBrush::paintLine	8 898 993 672	90.48
HairyBrush::paintParticle	7 146 890 813	72.67
KisRandomConstAccessor::moveTo	2 227 261 462	22.65
KisTiledRandomAccessor::moveTo	1 709 134 256	17.38
KisRandomConstAccessor::rawData	1 558 038 423	15.84

Tabuľka 9: Výsledky profilácie s anti-aliasingom

Pri implementácií Hairy brush štetca sme v prvých iteráciách tvorili masku štetca parametricky pomocou Gaussovej distribúcie. Pri rôznych nastaveniach štetca vznikali často krátke štetiny. Keďže pri vykresľovaní závisí priehľadnosť vykreslenej štetiny od jej dĺžky, štetiny zbytočne spomaľovali vykresľovanie a vôbec nemenili stopu štetca. Preto sme zaviedli po vytvorení tvaru štetca funkciu, ktorá vyraďovala štetiny na základe jej dĺžky. Štetiny sú alokované dynamicky, ich vyraďovanie bolo prevádzané ich odstránením. Tento krok sa po profilácií ukázal ako veľmi pomalý. Výkon sa podarilo zlepšiť pomocou implementácie časticového poolu, ktorý je odvodený od konceptu memory pool[9]. Prvý návrh tohto konceptu mal pracovať s dvoma vektormi, pričom jeden vektor obsahuje smerníky na aktívne častice a druhý vektor obsahuje vyradené smerníky na nepoužité častice. Metóde vykresľovania predáme zoznam smerníkov s aktívnymi časticami. Nakoniec bol koncept implementovaný pomocou príznaku aktívnej častice. Príznakom sme zlepšil výkon 1,5-krát. Vo finálnej implementácií nepoužívame parametrické generovanie masky, ale masku určuje digitálny obraz. Tento spôsob bol použiteľnejší pre umelcov. Vyraďovanie štetín vo finálnej implementácií prebieha ešte pred alokáciou pamäte, takže tento krok už nespomaľoval výkon.

Spray brush

Pre tento štetec sme testovali výkon konfigurácií, ktoré boli použité pri maľbe 39. Prvá konfigurácia používa len dve častice na jednu masku s premenlivou veľkosťou. Častice

[9]Memory pool koncept `http://www.boost.org/doc/libs/1_42_0/libs/pool/doc/concepts.html`

dopadajú do kruhovej oblasti s priemerom 17 pixlov, avšak parameter zmeny mierky, ktorý je nastavený na hodnotu 15% znižuje túto plochu na priemer 3 pixle. Typ častice je rasterizovaný polygón, konkrétne ide o kruh.

Test	Čas (v milisekundách)
Beziérová krivka	2,795
Náhodné čiary	12,971

Tabuľka 10: Rýchlosť štetca Spray brush s dvoma časticami

Štetec reaguje v reálnom čase, je ale pomalší so zvyšujúcim sa počtom častíc a ich veľkostí. V tabuľke 10 vidíme rýchlosť štetca pri použití dvoch častíc o veľkosť 7 pixlov. V tabuľke 11 vidíme konfiguráciu s 21 časticami na masku, ich veľkosť je 14 pixlov. Štetec sa spomalil takmer 6-násobne. Z profilovacích dát vidíme, že najviac času strávi procesor pri rasterizácií. Častice vykresľujeme pomocou funkcie KisPainter::fillPainterPath(), ktorú používa program Krita pri rasterizácii všeobecného polygónu. Túto funkciu sme optimalizovali, avšak stále patrí k funkciám, ktoré najviac spomaľujú odozvu štetca.

Test	Čas (v milisekundách)
Beziérová krivka	14,524
Náhodné čiary	72,398

Tabuľka 11: Rýchlosť štetca Spray brush s 21 časticami

Výsledky profilácie sú pre obe konfigurácie veľmi podobné, uvádzame výsledky profilácie pre nastavenie štetca s 21 časticami v tabuľke 12. Vidíme, že až 90% času trávi štetec rasterizáciou. QPainter::fillPath() je funkcia, ktorú voláme pri konštrukcií tvarov častíc. Vytvára cestu pomocou bodov a po uzavretí cesty je vyplnená zvolenou farbou.

Funkcia	Počet inštrukcií	Relatívny počet inštrukcií
Spraybrush::paint	26 647 894 028	95.88
SprayBrush::paintCircle	26 575 094 944	95.61
KisPainter::fillPainterPath	25 209 942 544	90.70
QPainter::fillPath	4 264 095 178	15.34

Tabuľka 12: Výsledky profilácie Spray brush s 21 časticami

Ďalšie konfigurácie testujú rýchlosť štetca Spray brush použitím jednoduchých častíc pre vykreslenie ako je Wu častica a textúra. Pri konfigurácií s textúrou nedochádza k rasterizácií, ale iba ku kompozícií častice. V tabuľke 13 vidíme rýchlosť štetca pre konfiguráciu s použitím Wu častíc. Počet častíc je zadaný relatívne k veľkosti štetca. 50% plochy štetca pokrývajú dopadajúce častice. Veľkosť štetca je 30 pixlov, pričom jeho veľkosť reaguje na tlak tabletu. Odozva štetca je dobrá, umožňuje vykresľovať veľké množstvo častíc.

Test	Čas (v milisekundách)
Beziérová krivka	3,663
Náhodné čiary	20,290

Tabuľka 13: Rýchlosť štetca Spray brush s 50% hustotou Wu častíc

Z profilovacích dát v tabuľke 14 vidíme, že najviac zamestnáva procesor kopírovanie farby. Pri výpočte alfa kanálu pixlov Wu častice kopírujeme užívateľom zvolenú farbu pre každú časticu. Funkciou KisRandomConstAccessor::moveTo pristupujeme k pamäti, do ktorej kopírujeme novú hodnotu pixlu. V štetci Spray brush používame opäť dlaždicový typ pamäte, pretože rozmer pomocného pixel bufferu vopred nepoznáme. V profilácii vidíme aj funkciu kompozície bitBlt pomocného pixel bufferu. Funkcia drand48 je používaná pri generácií pozícií pixlov a je veľmi často volaná.

Funkcia	Počet inštrukcií	Relatívny počet inštrukcií
SprayBrush::paint	5 527 574 151	75.16
KoColor::KoColor	1 002 215 715	57.02
KisRandomConstAccessor::moveTo	986 444 737	13.41
KisPainter::bitBlt	897 855 960	12.21
drand48	835 165 296	11.36

Tabuľka 14: Výsledky profilácie Spray brush s Wu časticami

Posledný typ testu pre tento štetec je test s textúrou. Zvolili sme textúru listu a bola aktivovaná náhodná transformácia pomocou HSV modelu. Štetec v tejto konfigurácií generoval dve častice na jednu masku. Odozva bola veľmi dobrá ako môžeme vidieť v tabuľke 15. Štetec reagoval rýchlo, pretože nedochádzalo k rasterizácií. Avšak namiesto rasterizácie dochádzalo k farebnej transformácii vstupnej masky.

Test	Čas (v milisekundách)
Beziérová krivka	551
Náhodné čiary	2,590

Tabuľka 15: Rýchlosť štetca Spray brush s textúrou

Funkcia	Počet inštrukcií	Relatívny počet inštrukcií
SprayBrush::paint	1 191 541 108	58.10
KisHSVAdjustment::transform	471 005 740	22.97
QImage::transformed	334 876 001	16.33
QPainter::drawImage	317 756 147	15.49

Tabuľka 16: Výsledky profilácie Spray brush s textúrou

Z profilovacích dát v tabuľke 16 vidíme, že najviac času trávi štetec farebnou trasformáciou textúr. Funkcia KisHSVAdjustment::transform() je volaná pre každý pixel textúry. Pomocou funkcie frameworku Qt QImage::transformed je textúra transformovaná podľa rotácie alebo zmeny mierky častice. Transformácia obrazu je pomalá, pretože je to opäť operácia pracujúca s každým pixlom vstupnej textúry. S transformáciou obrazu súvisí i funkcia QPainter::drawImage().

5 Záver

Navrhnuté typy štetcov som úspešne implementoval a integroval do programu Krita. Krita je komplexný nástroj pre spracovanie digitálneho obrazu so zameraním na digitálne maľovanie. Implementované štetce v ňom pracujú v reálnom čase s rýchlou odozvou pre obrazy s vysokým rozlíšením. Pomocou štetcov dokážeme vytvárať širokú škálu rôznych efektov a podarilo sa nám dosiahnuť pomerne realistickú simuláciu rôznych tradičných médií. Štetce dokážu simulovať maľbu uhľom, maľbu čínskeho štetca v štýle sumi-e i funkčnosť nástroja Airbrush. Každý implementovaný štetec poskytuje široké možnosti konfigurácie a umožňuje užívateľovi experimentovať s jednotlivými štetcami. Umelec si môže prispôsobiť štetce svojmu štýlu a vytvárať originálne maľby.

Táto diplomová práca bola motivovaná grantom programu Google Summer Of Code, ktorý som získal v roku 2008. V rámci tohto grantu som pracoval 3 mesiace na štetci Hairy brush a jeho integrácií do programu Krita. Jeho implementácia bola značne vylepšená vďaka tejto diplomovej práci. V roku 2009 som získal ďalší Google Summer Of Code grant, v rámci ktorého som pracoval opäť na programe Krita. Tento grant mi umožnil pracovať na iných subsystémoch programu Krita ako je užívateľské rozhranie a plátno. V roku 2010 som získal grant od komunity ľudí okolo programu Krita a pracoval som na celkovej optimalizácií programu. Znalosti z grantu a práce na optimalizáciách mi umožnili efektívnejšie využívať API programu Krita, vďaka čomu mohli byť štetce optimálne integrované.

Vývoj a proces integrácie prebiehal v období takmer dvoch rokov s časovými odstupmi. Vzniklo viacero štetcov a tie najpoužiteľnejšie sú popísané v tejto diplomovej práci. Program Krita je open-source projekt, do ktorého prispieva veľa programátorov a je v neustálom vývoji. Subsystém štetcov počas mojej práce bol niekoľkokrát zmenený a vylepšený tak, aby podporoval nové vlastnosti. Proces integrácie prebiehal dobre vďaka spolupráci s tímom programátorov Krita, ktorý pomáhal radami pri implementácií nového API. Integrované štetce podporujú všetky nové pridané vlastnosti a sú súčasťou tohto programu. Program je voľne dostupný v rôznych distribúciách Linuxu.

Štetce boli testované bežnými užívateľmi i skutočnými umelcami digitálneho umenia. Hlásené chyby štetcov boli opravené. Veľa testovania, rád a požiadavkov na štetce som získal od Samyho Langa, ktorý je umelec a prispel i maľbami, ktoré môžeme vidieť v prílohe práce. David Revoy, umelec zodpovedný za koncept art pre animovaný film Sintel od Blender Foundation a umelec Przemek Golab testovali štetce a prispeli maľbami v prílohe práce.

Štetce pri simulácií používajú modely časticových systémov. Ďalší vývoj by mohol smerovať k zložitejším modelom štetca. Zaujímavé by bolo porovnávanie výstupov štetcov s fyzikálne korektným modelom,s deformáciou štetín a so simuláciou dynamiky tekutín. Ďalšia zaujímavá oblasť výskumu je pokročilejšie miešanie farieb spomínaným

modelom Kubelka-Monk.

Vďaka diplomovej práci som sa oboznámil s vývojom open-source softvéru. Práca na tak rozsiahlom a komplexnom projekte ako je program Krita je dobrou skúškou zručností vývojára softvéru. Prínosná bola spolupráca s ostatnými programátormi programu Krita i spätná väzba od užívateľov, testerov a umelcov. Získal som veľa skúseností s algoritmami rastrovej grafiky a spracovania obrazu a skúsenosti s rôznym typom optimalizácii.

6 Referencie

[1] Baxter, Bill *DAB: interactive haptic painting with 3D virtual brushes*, International Conference on Computer Graphics and Interactive Techniques p. 461 - 468, 2005

[2] Elias, Hugo *Wu pixels*, `http://freespace.virgin.net/hugo.elias/graphics/x_wupixl.htm`, 12.4.2010

[3] Haeberli, Paul, *Paint By Numbers: Abstract Image Representations*, SIGGRAPH '90 Conference Proceedings , p.207-214, August 1990

[4] Haeberli, Paul, *Grafica Obscura*, `http://www.graficaobscura.com/`, 13.2. 2010

[5] Chan C., Akleman E. and Chen J., *Two Methods for Creating Chinese Painting*, Proceedings of Pacific Graphics'02, Beijing, P. R. China, Oct. 2002

[6] Jansson E., *Brush Painting Algorithms* Institutionen for datorteknik Göteborg, diplomová práca, 2004 `http://www.ce.chalmers.se/~uffe/xjobb/BrushPaintingAlgorithms.pdf`

[7] Porter Thomas, Duff Tom *Compositing Digital Images*, Computer Graphics Vol. 18, No 3, SIGGRAPH '84 Conference Proceedings, p. 253–259, July 1984

[8] Smith, Alvy Ray, *Digital Paint Systems: An Anecdotal and Historical Overview*, IEEE Annals of the History of Computing, p.4-26, Jun 2001

[9] Smith, Alvy Ray, *Biography*, `http://www.alvyray.com/`, 3.2. 2010

[10] Strassmann, Steve, *Hairy brushes*, Proceedings of the 13th annual conference on Computer graphics and interactive techniques, p.225-232, August 1986

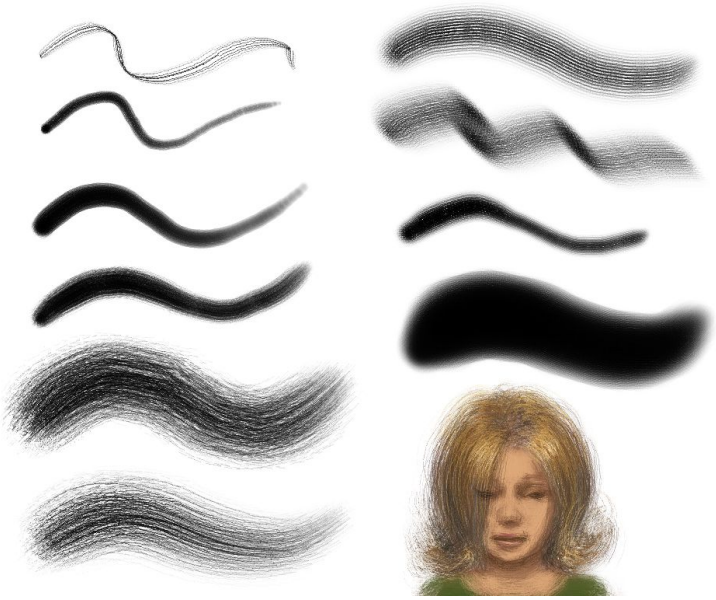
[11] Whitted, Turner, *Anti-aliased Line Drawing Using Brush Extrusion*, Computer Graphics 17, p. 151-156, July 1983

[12] *Matte Painting*, `http://en.wikipedia.org/wiki/Matte_painting`, 14.4.2010

[13] *Airbrush*, `http://en.wikipedia.org/wiki/Airbrush`, 14.4.2010

[14] Sam T. S. Wong , Howard Leung , Horace H. S. Ip, *Model-based analysis of Chinese calligraphy images*, Computer Vision and Image Understanding, v.109 n.1, p.69-85, January, 2008

A Kresby implementovanými štetcami

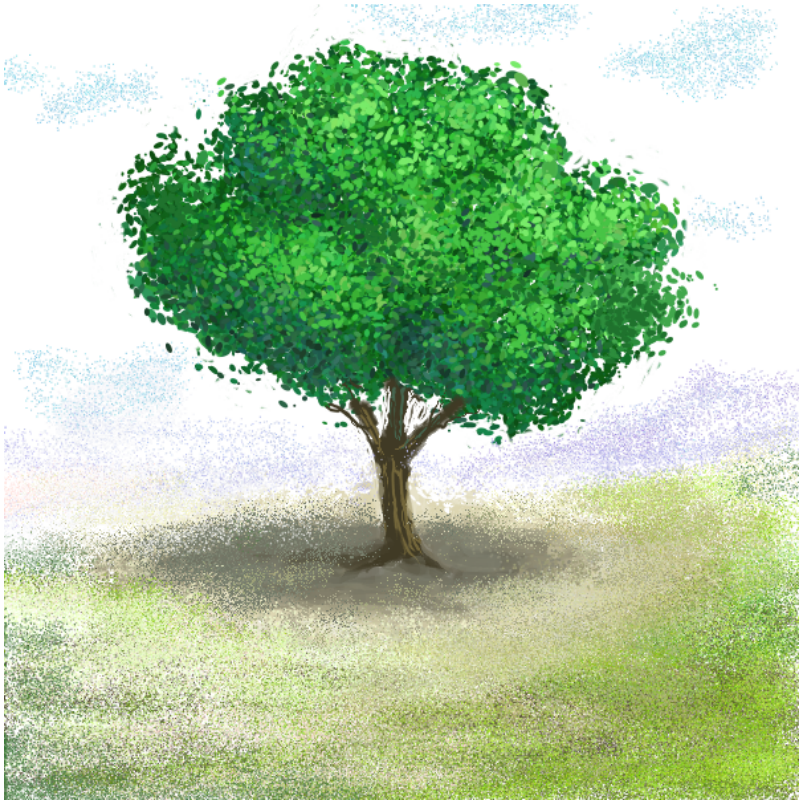


Obrázok 34: Demonštrácia štetca Hairy brush: Samy Lange - Girl

Obrázok 35: Demonštrácia štetca Hairy brush: Samy Lange - Rain

Obrázok 36: Demonštrácia Soft brush: David Revoy - maľba uhľom

Obrázok 37: Demonštrácia Hairy brush: David Revoy - Tiger



Obrázok 38: Spray brush: Samy Lange - Tree

Obrázok 39: Demonštrácia štetcov Soft brush a Spray brush: Przemek Golab - Watercolor